Part I

Fault Isolation Detection Expert (FIDEX)

Expert System Diagnostics
for a 30/20
Gigahertz Satellite Transponder

A final report summarizing research
on
Contract NAG3-923

completed at

The Univeristy of Akron
Electrical Engineering Department
Akron, Ohio 44325

for

NASA Lewis Research Center
21000 BrookPark Road
Cleveland, Ohio 44135

submitted by

Dr. John Durkin Associate Professor of Electrical Engineering
Richard Schlegelmilch Masters Student in Electrical Engineering
Donald Tallo Masters Student in Electrical Engineering

March 31, 1992

FIDEX

FAULT ISOLATION AND DIAGNOSIS EXPERT

An Expert System for Intelligent Computer Diagnostics of a Ka-Band Satellite Transponder System

ABSTRACT

The National Aeronautics and Space Administration (NASA), Lewis Research Center, in Cleveland Ohio, has recently completed the design of a Ka-band satellite transponder system, as part of the Advanced Communication Technology Satellite (ACTS) System. To enhance the reliability of this satellite, NASA funded The University of Akron to explore the application of an expert system to provide the transponder with an autonomous diagnosis capability. The result of this research was the development of a prototype diagnosis expert system called *FIDEX*©[1], *Fault Isolation and Diagnosis EXpert.*

FIDEX is a frame-based expert system that was developed in the NEXPERT Object™ development environment by Neuron Data, Inc. It is a MicroSoft® Windows™ version 3.0 application, and was designed to operate on an Intel i80386 based Personal Computer system.

[1]Antecedent to the publication of a thesis by Donald Tallo, an application has been made for the licence of Copyright. *FIDEX,* in the context of *Fault Isolation and Diagnostic Expert,* will be protected under that licence.

As a frame-based system, FIDEX uses hierarchical structures to represent such items as the satellite's: subsystems, components, sensors, and fault states. Its overall frame architecture integrates these hierarchical structures into a lattice that provides a flexible representation scheme and facilitates the maintenance of the knowledge-based system. To overcome limitations on the availability of sensor information, FIDEX uses an inexact reasoning technique based on the incrementally acquired evidence approach that was developed by Shortliffe during his MYCIN project. The system is also designed with a primitive learning ability through which it maintains a record of past diagnosis studies. This permits it to search first for those faults which are most likely to occur. And finally, FIDEX can detect abnormalities in the sensors which provide information on the transponder's performance. This ability is used to first rule out simple sensor malfunctions.

The overall design of the FIDEX system, with its generic structures and innovative features, makes it an applicable example for other types of diagnostic systems. This report discusses these aspects of FIDEX and summarizes the research involved in its development.

[redacted]

ACKNOWLEDGEMENTS

The authors would like to take this opportunity to acknowledge several individuals who have contributed to the completion of this research. We would first like to thank Edward Petrik for providing us the opportunity to pursue this research for NASA. His commitment to this research and his patience has been an invaluable asset to us. We would also like to thank Robert Kerczewski and James Svoboda for their time and patience. It has been their knowledge and all they have taught us that has made the development of the FIDEX system possible.

F I D E X

FAULT ISOLATION AND DIAGNOSIS EXPERT

An Expert System for Intelligent Computer Diagnostics of a Ka-Band Satellite Transponder System

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

LIST OF CODE SEGMENTS

CHAPTER I
INTRODUCTION

The satellite network of the United States supports both the commercial and military sectors by providing an effective world-wide communication network. The reliability of this network represents a strategic resource for this country and a critical concern for the *National Aeronautics and Space Administration (NASA).*

At present, the reliability of these satellite communication links are maintained through the intervention of ground terminal personnel. They use status information transmitted from the satellite to assess the possibility of system problems. Should these personnel suspect a problem with the satellite, a prescribed fault diagnosis/response strategy is followed. This process utilizes telemetry with mechanisms onboard the satellite to obtain required diagnostic information. Corrective measures are then communicated back to the switching mechanisms onboard the satellite which substitute redundant components.

This process can only occur when the satellite is within communication range; during fly-by. Otherwise, telemetry with the satellite is not possible. This limitation poses little problem for satellites in geosynchronous orbits because they are in constant communication with their controlling ground terminals. However, many satellites in the U.S. network are in asynchronous orbits. These satellites can be out of telemetry with their controlling ground stations for as much as 50% of the time. This situation represents a significant handicap in the maintenance of these communication links.

Current research in artificial intelligence (AI) has sought to resolve this problem by increasing the capabilities of the satellite's onboard diagnostic software. Since the mid 1980's, NASA has been investigating AI technology to develop a diagnostic expert system which could be placed onboard the spacecraft. If such a system could replicate the diagnostic tasks that are performed by the ground terminal personnel, it could provide the satellite with an autonomous diagnosis capability.

Success in this effort would offer the potential of significantly improving the reliability of satellite communication systems. Stephan [36] believes that achieving a high level of autonomy could allow the satellite to operate for months without ground contact. Such an enhancement could significantly reduce the cost of ground operations.

In the summer of 1988, NASA-Lewis Research Center funded The University of Akron to study the application of such a diagnosis expert system. This report discusses that study and the resulting prototype expert system called *FIDEX, Fault Isolation and Diagnosis Expert.*

1.1 Overview of the Application Area

NASA has recently completed the design of a Ka-band (30/20-GHz) communication satellite transponder. This transponder system is to be integrated within the Advanced Communication Technology Satellite (ACTS) System and deployed early in 1993. The ACTS transponder is a multiple channel repeater which relays microwave communication signals between highly localized ground terminals. All references to the transponder in this report are directed towards the components of the communication system that will reside onboard the satellite.

1.1.1 Overview of the ACTS Communication System

Figure 1.1 illustrates the ACTS communication system. The center figure, resembling a communication satellite, represents the ACTS transponder onboard the satellite. It is a typical multiple channel satellite transponder. Each channel of this system has an input and an output which are inter-linked through a matrix switch.

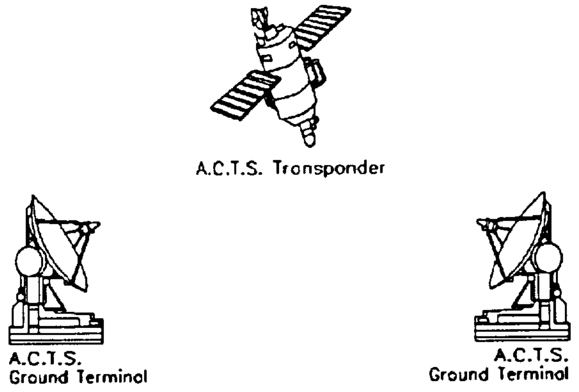

Figure 1.1: The ACTS Communication System

The transponder receives its inputs on a channel up-link from a ground terminal system. In the input channel, this signal is first amplified and then down-converted from the up-link frequency to an intermediate frequency (IF) signal. The IF signal is routed through a matrix switch to one of the output channels. There, the IF signal is up-converted to the broadcast frequency, amplified, and broadcast on a down-link to the proper destination ground terminal system. Only two channels are being implemented in the current phase of SITE; SITE Phase II. Therefore, this discussion is limited to that of a two channel system.

Associated with each channel through the transponder is a ground terminal system. They are represented in Figure 1.1 by the two satellite dishes. These stations are currently in a state of development similar to the transponder system. In fact, the actual units are not being used in the SITE testing of the transponder. Simulated ground stations have been substituted in their place until the entire system is ready for integration testing. Therefore, the following discussion is limited to that of ground stations in general, and not specific to the ACTS ground terminals.

Ground terminals represent the places of origin and destination for the signal transmitted through the transponder system. Respectively, these stations are called the originating and receiving ground terminals. In an originating ground station, digital information is encoded into a modulated signal. The modulated signal is up-converted to the up-link frequency, amplified and broadcast to the satellite. The satellite routes this signal as discussed above, and transmits it to the receiving ground terminal. In the receiving ground station, the down-link signal is amplified, down-converted, and demodulated back into digital information. In this manner, these three systems work together to provide cross-link communication between two geographically isolated places.

The remainder of this section explains the inner workings of the transponder system. This discussion, however, is limited to the workings of the SITE model of the transponder.

1.1.2 Overview of the ACTS Transponder System

Figure 1.2 shows a schematic representation of the ACTS transponder. At present, only two of the multiple channels are implemented in its design. However, this proof of concept design can easily be expanded to incorporate additional links as the system design progresses.

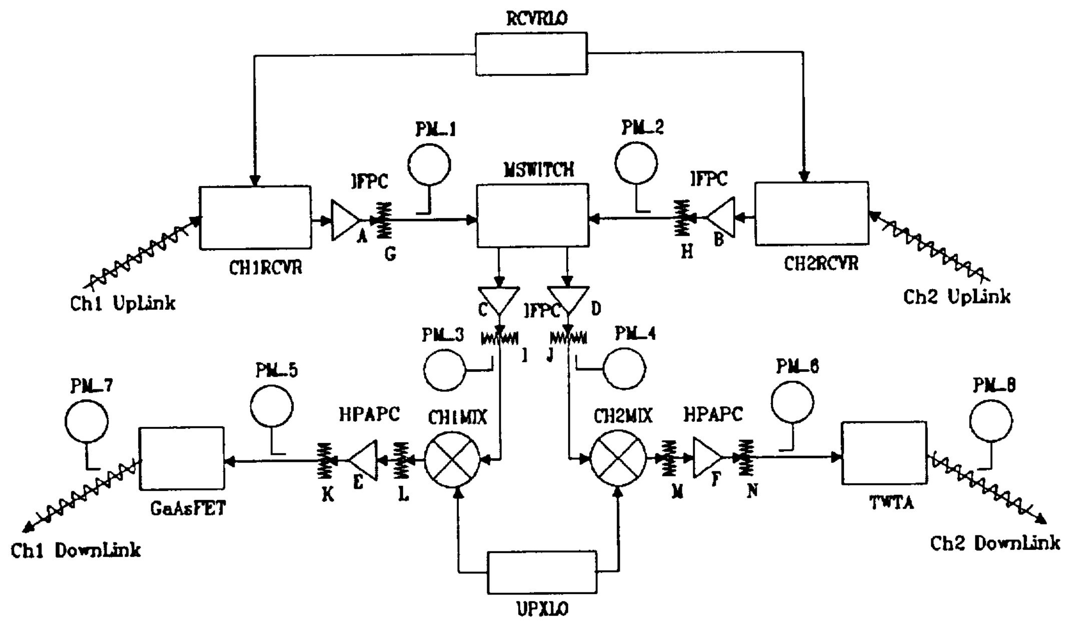

Figure 1.2: SITE Model of the ACTS Transponder System

At present, the design of this transponder is being evaluated within the System Integration, Test, and Evaluation (SITE) testbed at NASA-Lewis. The SITE laboratory[2] is used by NASA for validating designs and demonstrating the capabilities of satellite communications systems. This phase of development is valuable to NASA for refining the response of the various systems onboard the transponder. Another important aspect of SITE is the formulation of an understanding of these systems' fault response.

The first phase in the development of FIDEX was to study the diagnostic knowledge used by SITE personnel. This investigation began by studying the workings of the application area under normal conditions. This section is dedicated to summarizing that investigation. It is included to provide the background information on the ACTS transponder required by later discussions. For more detailed information on

[2]Space Electronics Division (SED), NASA-Lewis Research Center, Cleveland Ohio 44135

the transponder, please refer to the several papers listed in the SITE Related Publications appendix.

The transponder is the part of the communication system which routes signals between ground terminals. It consists of all elements of the communication system which will reside onboard the satellite. This system is a comparatively small part of the satellite system. Other major systems onboard the spacecraft include power systems, diagnostic systems, telemetric systems, etc. This discussion is limited only to the transponder. Therefore, references to the satellite are implicitly directed to the transponder system onboard.

Figure 1.2 shows the configuration of the transponder in its current SITE phase. As stated earlier, only two channels through the matrix switch are currently implemented. This will change as additional signal paths are added in future testing phases. However, the principles discussed here apply to any number of channels.

Table 1.1 provides a legend for the labels in Figure 1.2. The names in the table are functionally descriptive. They are not the names used by NASA personnel. The following discussion traces the functionality of each transponder element listed in this table.

Again, this section is only intended to provide background information on the workings of the transponder system. Therefore, each component's operation is discussed in a general manner regarding its function in system operation.

Table 1.1: Components of the ACTS Transponder System

	Major Transponder Components:
CH1RCVR	Channel 1 Receiver Unit
CH2RCVR	Channel 2 Receiver Unit
RCVRLO	Receiver Unit Local Oscillator
MSWITCH	Matrix Switch
CH1MIX	Channel 1 Up-Converter Mixer
CH2MIX	Channel 2 Up-Converter Mixer
UPXLO	Up-Converter Mixer Local Oscillator
GaAsFET	Gallium-Arsenide Field Effect Transistor Amplifier
TWTA	Traveling Wave Tube Amplifier
	Intermediate Frequency Power Control (IFPC) Amplifiers:
A	Channel 1 Matrix Switch Input IFPC Amplifier
B	Channel 2 Matrix Switch Input IFPC Amplifier
C	Channel 1 Up-converter Input IFPC Amplifier
D	Channel 2 Up-converter Input IFPC Amplifier
	Intermediate Frequency Power Control Attenuators:
G	Channel 1 Matrix Switch Input IFPC Attenuator
H	Channel 2 Matrix Switch Input IFPC Attenuator
I	Channel 1 Up-converter Input IFPC Attenuator
J	Channel 2 Up-converter Input IFPC Attenuator
	High Power Amplifier Input Power Control (HPAIPC) Amplifiers:
E	Channel 1 HPAIPC Driver Amplifier
F	Channel 2 HPAIPC Driver Amplifier
	High Power Amplifier Input Power Control Attenuators:
K	Channel 1 High Power Amplifier Input Attenuator
L	Channel 1 HPAIPC Driver Input Attenuator
M	Channel 2 HPAIPC Driver Input Attenuator
N	Channel 2 High Power Amplifier Input Attenuator
	Signal Power Level Sensors:
PM_1	Channel 1 Matrix Switch Input Signal Power Level Sensor
PM_2	Channel 2 Matrix Switch Input Signal Power Level Sensor
PM_3	Channel 1 Up-converter Input Signal Power Level Sensor
PM_4	Channel 2 Up-converter Input Signal Power Level Sensor
PM_5	Channel 1 HPA Input Signal Power Level Sensor
PM_6	Channel 2 HPA Input Signal Power Level Sensor
PM_7	Channel 1 HPA Output Signal Power Level Sensor
PM_8	Channel 2 HPA Output Signal Power Level Sensor

Transponder System Control and Monitoring

The first components of interest are those designated as *PM_1* through *PM_8* in Figure 1.2. These components represent the transponder signal power level sensors which are present in the SITE Phase II model. These signal power level sensors report the power level of the transponder signal at various key locations throughout the system. All sensor readings are made available as diagnostic information via two sources. First, each sensor has a digital display which is visible on the transponder system control panel. This display offers a visual means of obtaining power meter readings.

The second source for accessing sensory information is accomplished through an interface with a specialized computer called the Experiment Control and Monitoring (EC&M) Computer. This computer is discussed in greater detail later in this section.

Experiment control is provided by the adjustment of the attenuators within the transponder. These attenuators can be either controlled manually or via an interface with the EC&M computer. Additionally, these attenuator settings are monitored by the EC&M and can be reported by either visual displays or interfacing with this computer. The power meters are also interfaced with the EC&M computer. This interface provides computer access to signal power level readings at the sensor locations.

With this understanding of transponder control and monitoring, the workings of this system can be discussed on a component by component basis. Again, this discussion is based upon the workings of the transponder system during testing and evaluation experiments.

The two channels of the transponder system are symmetrical about the matrix switch. Each channel has both an input and an output. A channel's input consists of all components responsible for receiving an up-link broadcast and preparing it for input to the matrix switch. A channel's output consists of all components responsible for preparing the signal for broadcast on the down-link.

Uplink Receivers and IF Frequency Conversion

Channel 1 and 2 input channels are symmetrical about the matrix switch and can be discussed together. They receive up-link signals from their corresponding ground terminal system. This input signal is the modulated data stream being broadcast at 30 GigaHertz (GHz) from the ground terminals. The 30-GHz Low Noise Receiver units on the channel 1 and 2 inputs, *CH1RCVR* and *CH2RCVR* in Figure 1.2, receive the up-link signal at a very low signal power level. The receiver units must provide the necessary amplification and down-convert the signal frequency to 2.5-GHz, the Intermediate Frequency (IF) level.

This frequency down-conversion is accomplished via mixing with a local oscillator (LO) unit. Associated with both receiver units is the Receiver Local Oscillator, shown as component *RCVRLO* in Figure 1.2. This LO unit provides the 2.5-GHz reference necessary for down-conversion to the IF frequency.

After down-conversion, the IF signal power levels are controlled by components *A*, *B*, *G*, and *H*. The Intermediate Frequency Power Control (IFPC) Amplifiers, components *A* and *B*, provide a constant 43-dB amplification to the IF signal. This high gain is compensated for by the IFPC Attenuators, components *G* and *H*. These attenuators are under control of the EC&M Computer and are incrementally adjustable in 1-dB steps. This control allows the IF signal power level to be maintained to within 1-dB of its nominal level before input to the matrix switch. The IF signal power levels at the input to the matrix switch are monitored by the IF Power Level Sensors, *PM_1* and *PM_2*, and reported to the EC&M computer.

Ford Microwave Matrix Switch

Interconnectivity between channel inputs and outputs is provided by the Microwave Matrix Switch, component *MSWITCH* in Figure 1.2. This matrix switch is a switching unit having multiple input and output channels. The internal switching mechanisms provide cross point connections for a full permutation of signal paths through the switch. However, in the transponder's current phase of development, only two channels are implemented, channel 1 and channel 2. Consequently, only two of the many channels of the matrix switch are in use. This provides a total of four possible paths through the matrix switch. Table 1.2 shows the permutations of paths through the matrix switch.

Table 1.2: Permutations of Signal Paths Through Matrix Switch

Signal Path:	**Switch Interconnectivity:**
PATH11	Channel 1 Input -- Channel 1 Output
PATH12	Channel 1 Input -- Channel 2 Output
PATH21	Channel 2 Input -- Channel 1 Output
PATH22	Channel 2 Input -- Channel 2 Output

Inherent to the switching mechanisms, internal to the matrix switch and dependant upon signal path, is a certain degree of attenuation to the IF signal. Consequently, after signal path switching, the IF signal power level must be adjusted to maintain a proper signal strength. This signal power level control is affected by a second set of IFPC units, components *C*, *D*, *I*, and *J* in Figure 1.2. The IFPC Amplifiers, components *C* and *D*, again provide a constant 45-dB amplification to this signal, allowing the IFPC Attenuators, components *P* and *Q*, to provide 1-dB step control over the IF signal strength. The IF signal power levels are monitored and reported to the EC&M by IF Signal Power Level Sensors *PM_3* and *PM_4*.

The channel outputs of the transponder system are responsible for preparing the IF signal for broadcasting on the down-link to the ground terminal systems. After switching, the transponder IF signal is considered to be in the output channel of its path through the transponder system. Again, the channel outputs of the transponder system are symmetrical about the matrix switch. The only exception to this symmetry occurs at the high power broadcast amplifiers before down-link. This exception is discussed later.

IF/Downlink Frequency Conversion

At this point in the component discussion, the signal has been directed through the matrix switch to its proper output channel. It is now ready to be prepared for transmission to the ground terminals. The first step in this preparation of the signal for broadcast is the frequency up-conversion of the 2.5-GHz IF signal to the 20-GHz broadcast frequency. This is accomplished by components *CH1MIX* and *CH2MIX*, which are the Transponder System Up-converter Multiplexers. These units combine the 2.5-GHz IF signal with a 20-GHz reference signal provided by the Transponder System Up-converter Local Oscillator, component *UPXLO* in Figure 1.2.

After mixing, the signal power levels must be adjusted before input to the high power broadcast amplifier units, components *GaASFET* and *TWTA*. This is accomplished by the High Power Amplifier Input Power Control (HPAIPC) units which follow the multiplexers, components *E* through *N*. The first HPAIPC attenuators in the output channel, components *L* and *M*, are fixed devices which provide a constant attenuation. Next, the HPAIPC Driver Amplifiers, components *E* and *F*, amplify the signal between

25 to 31-dB. The subsequent attenuators, HPAIPC attenuators, components *K* and *N*, are rotary vane pin diode attenuators which are continuously variable. This continuity in their adjustment allows the EC&M computer to have precise control of the signal power level on input to the high power broadcast amplifier units. The signal power levels at the input to the broadcast amplifiers are monitored and reported to the EC&M Computer via power sensors *PM_5* and *PM_6*.

The two high power amplifier units, now amplify the signal for broadcast on the down-link to the ground terminals. This is the point where the symmetry of the output channels fails. These amplifiers perform similar functions in the operation of the transponder system. However, they are distinctly different units and must be discussed separately.

Gallium Arsenide (GaAs) FET Amplifier

Component *GaAsFET*, the 20-GHz Solid State Amplifier, at the output of channel 1, is a Gallium Arsenide Field Effect Transistor (GaAs FET) amplifier unit. This amplifier unit can be configured to operate at various set-points along its Input vs. Output (I/O) characteristic curve. By establishing a set-point, this amplifier can be configured to operate in one of three different modes; in a linear mode, in 1-dB compression or in a saturation mode. The following discussion helps to explain the multiple set-points for operation of an amplifier unit.

As for any amplifier, the gain of the GaAs FET amplifier can be plotted as the magnitude of its output power level versus the magnitude of its input power level. This plot is often called the characteristic curve of the amplifier. Figure 1.3 shows a typical characteristic curve for a GaAs FET Amplifier. This figure is only intended to provide a conceptual understanding of the linear, compression and saturation ranges of amplifier characteristic curves. It is not scaled to provide characteristic data about the GaAs FET amplifier specifically.

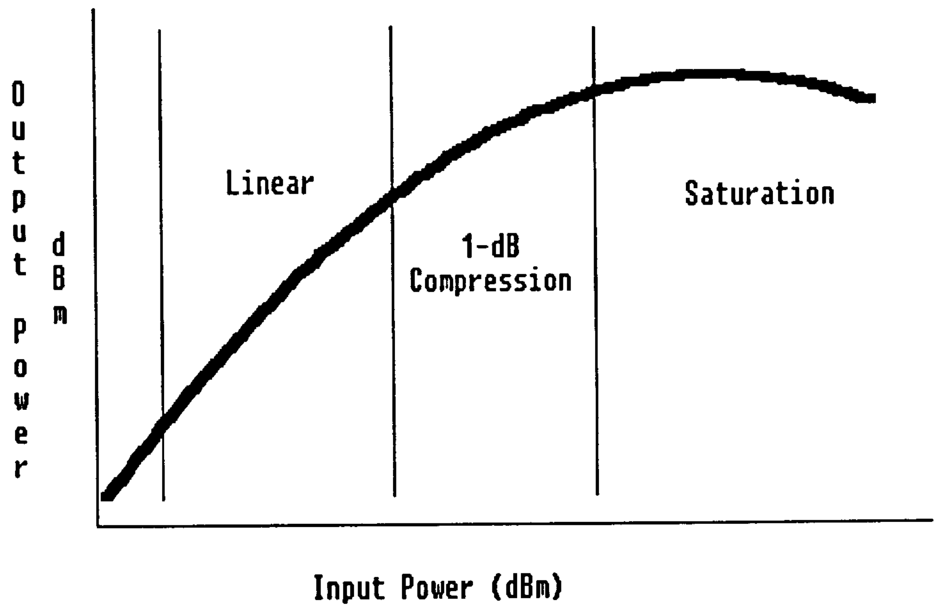

Figure 1.3: Characteristics of a GaAs FET Amplifier

This curve shows the non-linear relationship between the power at the output of an amplifier for a given input power level. Furthermore, this curve shows that the behavior of an amplifier is not consistent over the entire range of inputs. However, some generalizations can be made about amplifier behavior over specific regions of the curve.

Although the characteristic curve is never actually linear, the lower end of the input power scale exhibits a behavior which approximates a linear relationship. The range of input power levels which produce this nearly linear characteristic is therefore called the Linear Range. Figure 1.3 shows this linear behavior in the range marked

"Linear." Notice that over this range, the characteristics of the amplifier could be approximated by a straight line.

As the input power level is increased beyond this "Linear" range, the nonlinear parameters of an amplifier begin to become more pronounced. The characteristic curve begins to lose its linearity and compresses to a line of zero slope. Additionally, this compression is not constant and increases proportionally with the input power level. However, over a specific band of input power levels, the compression of the curve can be approximated as a 1-dB compression ratio. This range is therefore called the "1-dB Compression" range and is noted in Figure 1.3.

As the magnitude of the input power level is increased beyond this 1-dB compression range, the compression of the characteristic curve becomes very pronounced. Its slope begins to approach zero rapidly, as the amplifier unit approaches saturation. The band of all input power levels above the 1-dB compression range exhibit this behavior. Therefore, this band of input power magnitudes is called the "Saturation" range and is also indicated on Figure 1.3.

The configuration of the broadcast amplifier has a direct effect on the quality or accuracy of the communication signal. Ideally, an amplifier operates most efficiently near its saturated region. Near this range, the amplifier is providing the highest possible gain to an input signal. However, as the operating point of the amplifier approaches saturation, the amount of noise induced in the transmitted signal becomes greater. This noise is induced by the saturated amplifier as an increase in the harmonic content of the signal. Data bit errors begin to occur when this noise rises above a certain threshold limit.

To prevent any loss of data in the modulated signal being transmitted through the satellite transponder, a set-point should be chosen in the linear region of operation. However, the gain characteristics of the amplifier in the linear range do not provide for efficient amplification of the broadcast signal. Consequently, a trade-off must be made between the broadcast power required for efficient transmission and the linearity required for accurate transmission of data through the transponder.

Multi-Mode Traveling Wave Tube Amplifier (TWTA)

Component *TWTA*, the 20-GHz Multi-Mode Traveling Wave Tube Amplifier at the output of channel 2, is a Traveling Wave Tube Amplifier (TWTA). This amplifier can be configured to operate in various power modes. This multi-mode behavior allows the amplifier to operate along several characteristic curves; as opposed to a single curve like the GaAs FET. Specifically, the Multi-Mode TWTA can operate along one of three characteristic curves; each corresponding to one power mode of the TWTA. Figure 1.4 shows typical power characteristics for a Multi-Mode TWTA.

The Low Power Mode is designed for optimal efficiency in broadcast power. The gain of this mode is limited. However, under normal atmospheric conditions, this mode of operation should provide sufficient broadcast power for communication with the ground terminals.

However, adverse atmospheric conditions often require a satellite to step up its broadcast power to overcome interferences. This TWTA can be stepped up to a Medium Power Mode to provide this compensation. The Medium mode of operation provides a higher gain and therefore an increase in the strength of the broadcast signal. The consequence is a greater power consumption by the amplifier unit.

Similarly, should additional gain be required beyond that of the Medium Power Mode, a third configuration of the TWTA modes is available. The High Power Mode provides a very high gain to compensate for extremely strong atmospheric interferences, such as rain.

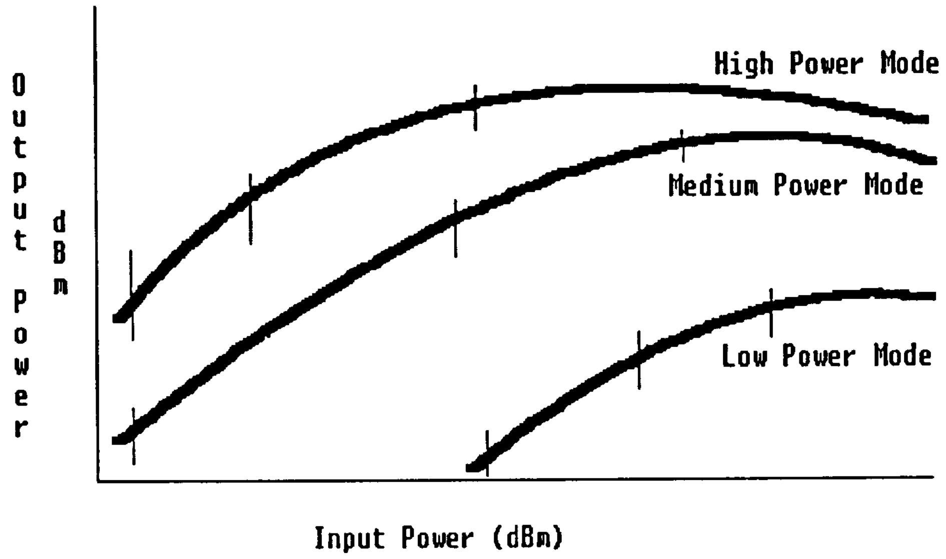

Figure 1.4: Characteristics of a Traveling Wave Tube Amplifier

In addition to having the multi-mode capabilities, the TWTA can also be configured to operate at various set-points along each of its characteristic curves. Similar to the GaAs FET, the TWTA can operate linearly, in compression or in saturation. It has three operating points in each power mode; resulting in a total of nine possible configurations for the TWTA.

The final components in the transmission paths through the transponder system are the transponder signal power level sensors *PM_7* and *PM_8*. After amplification by one of the broadcast amplifiers, the power levels of the down-link signal is reported to the EC&M via these sensors. These sensor readings indicate the output power of the transponder system, as the signal is transmitted to one of the ground terminals. In the current phase of development, transmission of the signal to and from the ground terminals is simulated by a direct wave guide link.

In summary, the transponder system consists of many components responsible for receiving, routing and broadcasting a communication signal between the ground terminals. A signal is received on a channel up-link from its originating ground terminal system. This signal is then down-converted in frequency to an intermediate frequency and routed through a matrix switch. This matrix switch is a switching network which channels a signal through one of four possible paths. After switching, the signal is up-converted in frequency and amplified by high power amplifiers and then broadcast on a channel down-link to its destination ground terminal.

The next section looks into the computer control and monitoring performed by the various computer systems associated with the transponder and ground terminal.

1.1.3 Overview of Computer Control & Monitoring

The final topic of discussion in this overview is the computer network responsible for the control and monitoring of the satellite transponder and ground terminal systems. Figure 1.5 is an interface diagram which shows this network.

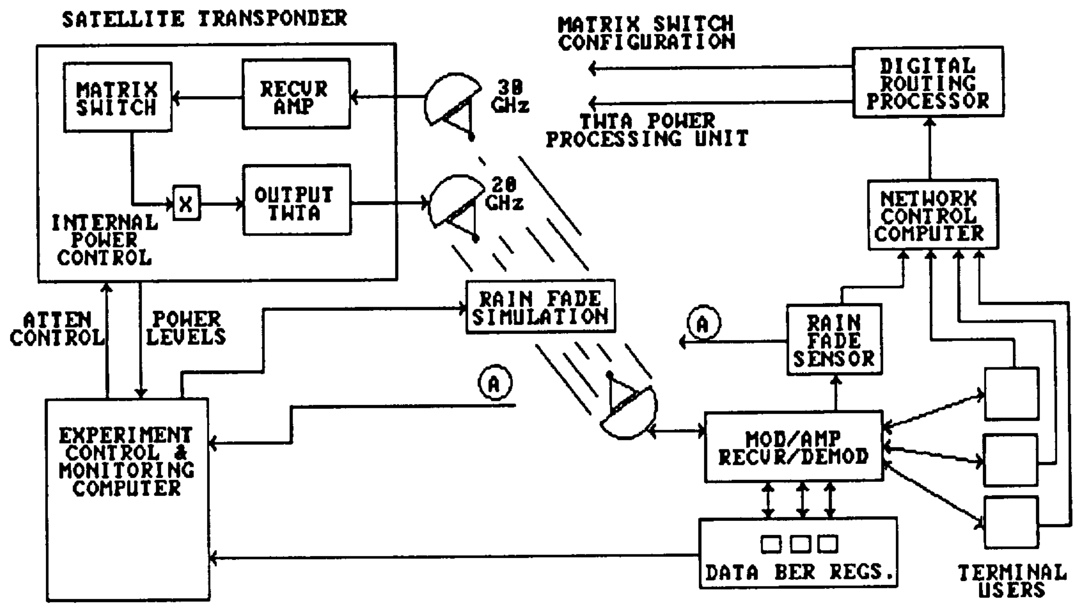

Figure 1.5: Computer Control and Monitoring Interface

Digital Ground Terminal

The first computer network of interest is a group of digital processors located in the ground terminal system. This group is collectively called the Digital Ground Terminal. It is comprised of the three simulated user terminals and the Data Bit Error Rate Registers. Again, as stated earlier, these processors are only present to simulate the existence of users. The actual ground terminal computer is currently under independent development, therefore, this discussion is limited to the operation of the simulated user processors.

Located in the bottom right hand corner of Figure 1.5 are three blocks labeled as User Terminals. In the current stage of development for this system, only three system users are being simulated. It is these three processors which are used to simulate the users. Each user terminal originates an individual data set to be transmitted through the communication system. During experimentation, large data sets are required for transmission. To accomplish this generation of a large data set, each user terminal generates a pseudo-random stream of bits. This data it then transmitted digitally to the signal processing components of the ground terminal.

The signal processing components transmit this data through the satellite transponder system and then returns it digitally to the appropriate users. After the transmitted data is received at the appropriate user terminal, the data stream is checked for bit errors. This is accomplished by an Exclusive-OR (XOR) with the bits of the original data stream. The total of "Missed Bits" is calculated and a Bit Error Rate (BER) determined. This BER is simply the ratio of missed bits to the number of bits transmitted.

This BER provides useful information about the performance of the communication under the experimental conditions. Therefore, the results of error checking for each of the users are stored in the three BER Registers shown in Figure 1.5. Since these registers are also digital components involved in the generation and evaluation of digital data, they are included in the network as part of the Digital Ground Terminal. In all, the components of this network, collectively called the Digital Ground Terminal, are responsible for the generation and evaluation of data to be transmitted during the testing and evaluation of the transponder system.

Network Control Computer

The next computer of interest is the block in Figure 1.5 labeled as the Network Control Computer (NCC). This computer is responsible for evaluating information from the Ground Terminal System and appropriate control of the Transponder System. Its primary task is two fold. First, it must control the routing of the transmitted signal through the transponder system. And second, it must compensate for atmospheric disturbances by controlling the output signal power level of the satellite broadcast amplifiers.

The NCC receives input from the simulated ground terminal users for the proper configuration of the matrix switch in the transponder system. Then, it configures the matrix switch to assure proper routing of data transmitted through the transponder. The NCC also is responsible for controlling the output power levels of the transponder's broadcast signal. The primary reason for this output power level control is compensation for power losses which result from atmospheric disturbances, commonly called "Rain Fade." Located in the signal processing portion of the ground terminal system is a rain fade sensor. This sensor detects power attenuation caused by the rain fade simulator. When the down-link signal power level, reported by the rain fade sensor, falls below a certain threshold, the NCC directs the power processing unit of the Multi Mode TWTA to compensate for this attenuation by changing power modes. Conversely, to save power, if the rain fade sensor reports a decrease in attenuation, the NCC instructs the TWTA's power processing unit to change to a lower power mode to compensate.

Experiment Control & Monitoring Computer

In the current phase of integrating, testing, and evaluating the transponder and its associated ground terminals, all operation is controlled and evaluated by the EC&M Computer. This computer is responsible for controlling all adjustable attenuators and the reporting of all sensory information.

Several attenuator settings can be controlled and reported by this computer. These settings are controlled by an analog voltage generated via an interface with the EC&M. The magnitude of this control voltage is also available for reporting the current levels of attenuation in these settings.

In addition to controlling the transponder system signal power levels, the EC&M also provides a means of monitoring and evaluating the condition of the transponder and associated ground terminals by reporting sensory information. There are fifteen sensory inputs to the EC&M which report the readings of the nine transponder signal power level sensors and six data stream bit error rates.

In the current phase of development of FIDEX, there is no direct interface between this EC&M computer and the diagnostic process. Currently, this information is input via user interrogation. However, the availability of this data would lend to a future implementation of an interface between this expert system and the EC&M computer. This concept is discussed in greater detail, when appropriate, later in this report.

1.2 Project Definition

The goal of this research project was to investigate the possibility of representing the knowledge gained during SITE in a diagnostic expert system. Such a study would then help to lay groundwork for a future system capable of providing the transponder with autonomous diagnosis capability. The research for this project progressed according to several key developmental phases:

1. *Domain Analysis:* Study the operation of the application system under both normal and abnormal conditions

2. *Knowledge Acquisition:* Study and organize the knowledge used by the domain experts who perform fault diagnostics on application system

3. *Knowledge Representation:* Design a scheme to model the application system and represent the knowledge required to detect, isolate, and diagnose its fault states

4. *Response Strategy Definition:* Establish response strategies and procedures for all fault states

5. *Prototype Development:* Develop, test, and modify a series of evolutionary prototype diagnostic expert systems

6. *Requirements Definition:* Define the overall specifications for the final deliverable diagnostic expert system

7. *Final Development:* Design, encode, integrate, test, and document the final deliverable expert system

8. *Life Cycle Analysis:* Define and specify a maintenance schedule for the deliverable diagnostic expert system

During these phases of development, several problems were encountered which reshaped the requirements of the project. Three problems of particular interest resulted from the evolutionary state of the ACTS transponder system. The requirements which these difficulties added to the project, and their solutions, highlight the major strengths of this expert system.

The first of these difficulties became evident during domain analysis. The expert system was constrained to work with limited information on the operational condition of the transponder. Specifically, there were only a few sensors available to provide information on the response of the transponder system. This information was limited to the signal power level sensors, indicated in Figure 1.2 as *PM_1* through *PM_8*, and a few bit-error-rate (BER) registers. This limited information was not completely adequate for assessing the condition of the transponder. In short, the sensors in the transponder were sparse in number, compared to the other components of the transponder system. Therefore, the isolation of a fault to a specific component based upon sensory information alone was not possible. This limitation was termed the *Sparse Sensor Problem*.

This problem also placed a high premium on the reliability of sensory information. Inconsistent or erroneous readings could render the expert system inoperable. Therefore, a method for resolving conflicts in sensory data was needed.

A second problem was encountered during knowledge acquisition. A prerequisite for the development of an expert system is an extensive understanding of the application area. In a diagnostic application, this requirement dictates that the potential fault states of the system be well known. However, the ACTS transponder was still under evaluation, and a complete understanding of its fault response had yet to be formulated. This fact constrained the investigators to work with limited diagnostic knowledge. Without a clear definition of the transponder's fault response, explicit diagnostic rules were not possible. Therefore, the expert system was prescribed to work with abstract, as well as concrete diagnostic knowledge.

The final problem was also a result of the evolutionary state of the transponder system. The problem was that changes in the design of the system were always possible. These changes could range from modifications to design specifications, or even the addition of new modules. This situation made it difficult to develop a robust diagnostic agenda.

Faced with these problems, the goal of this project changed more towards a study effort. Emphasis was placed on the development of techniques that would overcome these problems and permit the expert system to reason intelligently with only limited information. The system's knowledge needed to be structured such that any change in the design of the transponder could easily be reflected in the structure of the expert system. All of these requirements placed a premium on the design of knowledge representation techniques and reasoning methods that were general and flexible.

The result of this effort was the development of a prototype diagnostic expert system called FIDEX, Fault Isolation and Diagnosis EXpert. This project demonstrated the feasibility of developing an intelligent computer diagnostic system not only for the ACTS transponder, but for space systems in general.

1.3 General Approach to Solution

The general approach taken in the development of this project followed the problem-solving approach used by the ground personnel who perform satellite diagnostics. This strategy was termed the *Modular Approach to Diagnostics*. In general, it follows the four tasks of:

1. *Fault Detection:* Monitor the response of the transponder to determine whether it is functioning properly or not
2. *Fault Isolation:* Narrow the range of suspected components to the smallest possible group
3. *Fault Diagnosis:* Investigate the precise nature of the misbehavior and determine the component causing it
4. *Fault Response:* Respond to the diagnosis in a robust and intelligent manner

Fault Detection

The purpose of the first task, *Fault Detection*, is to detect any misbehavior in the transponder performance. This task involves the analysis of current sensor information to ascribe qualitative descriptions to each sensor's reading; either *"GOOD"* or *"BAD."* These descriptions are based on whether the data reported by a sensor exceeds a tolerance figure centered on its nominal or expected value. Sensor readings which are within tolerance receive a *"GOOD"* description, and those which exceeded their tolerance range are labeled as "*BAD.*" The detection of a fault is based upon establishing a "*BAD*" reading on any sensor. This indicates that a misbehavior exists in the transponder system and causes the next task to begin.

Fault Isolation and Sensor Validation

The second task in this approach is *Fault Isolation*. Its purpose is to isolate the suspected fault to the smallest possible group of components in the transponder. This is accomplished through a principle known as *Error Propagation*. This principle states that the observable symptoms of a misbehavior in a component propagates through all subsequent sensors in a signal path. The source of such a misbehavior can thus be concluded to lie in that signal path, prior to the detection of the misbehavior, and subsequent to the last sensor indicating a proper signal response.

To implement this, the isolation task considers the qualitative description of all sensor readings ascribe by the detection phase. It locates a sensor reporting a "*GOOD*" reading which is followed by a "*BAD*" reading. However, because of the sparse sensor limitations, this approach can only isolate the source of the misbehavior to the group of components between these two sensors. For the purposes of this project, these groups of components are termed *SubSystems*, and are defined as the groups of components bounded signal power level sensors.

The fault isolation task relies heavily upon the integrity of the data reported by the sensors. Should any sensor report erroneous data, this task will fail to reach a valid conclusion. Therefore, a subordinate *Sensor Validation* task was added to this diagnostic phase.

The sub-task of sensor validation is designed to identify the possibility of a faulty sensor. This ability permits the FIDEX system to avoid the search for a non-existing transponder fault. Sensor validation is also based on error propagation; however, in a slightly different fashion. Again, a signal producing a "*BAD*" sensor reading at one point in the transponder should result in a "*BAD*" reading on all subsequent sensors in that signal path. This task identifies the possibility of a faulted sensor if a "*GOOD*" reading instead is found.

In either case, the purpose of isolation is to identify the subsystem containing the component causing the misbehavior. If this misbehavior is the result of a component failure, the subsystem identified by its input and output sensor readings is flagged as isolated. However, if the detected "*BAD*" sensor reading is the result of a faulty sensor, isolation flags the sensory components as the isolated subsystem. Once the source of the fault is isolated, the next task is initiated.

Fault Diagnosis

The third task, *Fault Diagnosis*, involves consulting a community of diagnostic expert systems. Each system is designed to address the problems of a specific subsystem within the transponder. Determining the appropriate diagnostic expert to be consulted is the final task of the isolation phase.

These specialized diagnostic systems use knowledge which is rule-based and backward chaining in nature. The hypotheses for these rules represent the potential faults in the isolated subsystem. The order in which they are placed on the agenda is based on the history of the fault states. Maintaining this history permits FIDEX to pursue the most likely problems first.

Each diagnostic system was also designed with an ability to perform inexact reasoning. This was done to overcome problems which resulted from limited information about the transponder's performance. Such an ability was important in that the FIDEX system would often need to make a "guess" at the most likely fault state.

The inexact reasoning technique chosen for this project was based on the certainty theory given by Shortliffe [34]. It relies upon establishing incremental measures of belief or disbelief in rule conclusions. These two factors are then used to establish an overall confidence when a conclusion is supported by multiple rules.

Fault Response

The final task is *Fault Response*. The present strategy for fault response is to provide recommendations for reconfiguring the components or sensors. Plans are to include the capability to reconsider fault diagnosis if the recommended action was ineffective. FIDEX would retain its past diagnosis, including recommendations, and reconsider the problem with information made available following the corrections to the transponder.

The remainder of this paper discusses the workings of the FIDEX system. It demonstrates the techniques discussed above, and by example, show their application to other types of diagnostic systems.

CHAPTER II
DEVELOPMENT ENVIRONMENT

The long term objective for FIDEX was to permit it to acquire data on the transponder through the satellite's onboard data acquisition systems. However, during its development of FIDEX it was decided to acquire this data interactively from the user. Therefore, a user interface between NEXPERT Object™ and ToolBook™ was developed. These software packages operate in the MicroSoft® Windows™ environment. This permits them to interact and communicate through dynamic data exchange (DDE).

Being a Windows™ application, the interface is highly menu driven. The user enters information about the condition of the transponder through various forms and prompts. This data is then dynamically transferred to the NEXPERT™ application where it is evaluated. The interface also allows NEXPERT™ to prompt the user for information as it is required during the diagnostic process.

The FIDEX system needed to be designed in a fashion that would allow it to incorporate changes to the transponder easily. Therefore, a frame-based approach was taken for knowledge representation. The system was also required to operate on an i80386 machine. The NEXPERT Object™ development environment, by Neuron Data, Inc., was chosen as the development environment for the FIDEX system.

NEXPERT™ permits an object-oriented style of programming within class/subclass-object/subobject hierarchies. It includes message passing through active facets. It allows the encoding of general rules that scan frame hierarchies. It also permits access to database information contained in a variety of database formats. It can be linked with and execute external programs. As a MicroSoft® Windows™ application, it supports dynamic data exchange (DDE) and external message passing through dynamic link libraries (DLLs). All of these features were important in the design of FIDEX. Table 2.1 shows the basic system requirements for using NEXPERT Object™.

Table 2.1: NEXPERT Object™ System Requirements

- IBM PC Model 80, or compatible i80386 machine, with 1024-KBytes of base memory plus a minimum of 2048-KBytes extended memory.
- MicroSoft™ Windows™ version 3.0 or later.
- Enhanced Graphics Array (EGA) or VGA color graphics adaptor with a minimum of 64-KBytes graphics memory.
- 1.2-MByte or 1.44-MByte floppy disk drive.
- Hard disk with a minimum of 6-MBytes available disk space.
- Programmable bi-directional parallel port.

CHAPTER III
KNOWLEDGE ENGINEERING

The diagnostic knowledge of FIDEX is represented using both frame-based and rule-based techniques. This section discusses the structure of this hybrid framework. This is required to provide a background for discussions in subsequent chapters which describe the actual implementation of FIDEX in the syntax of NEXPERT Object™.

3.1 Frame-Based Architecture

The expert system needed to be designed such that it would easily allow the incorporation of changes to the transponder. Therefore, it was decided that a frame-based approach for knowledge representation would be appropriate. Frame hierarchies were developed to represent the transponder's components, subsystems, sensors and fault states. These hierarchies were interconnected into a network to enrich the overall knowledge representation structure.

3.1.1 Structure of Components Class Hierarchy

A frame hierarchy was created to provide a clear and efficient representation of all components in the transponder. Figure 3.1 shows this structure called the *Components Class Hierarchy*. This figure illustrates a convention that is maintained throughout in this report. Circles represent class frames and triangles represent object frames. Lines indicate links between frames, with the arrows indicating the direction of inheritance.

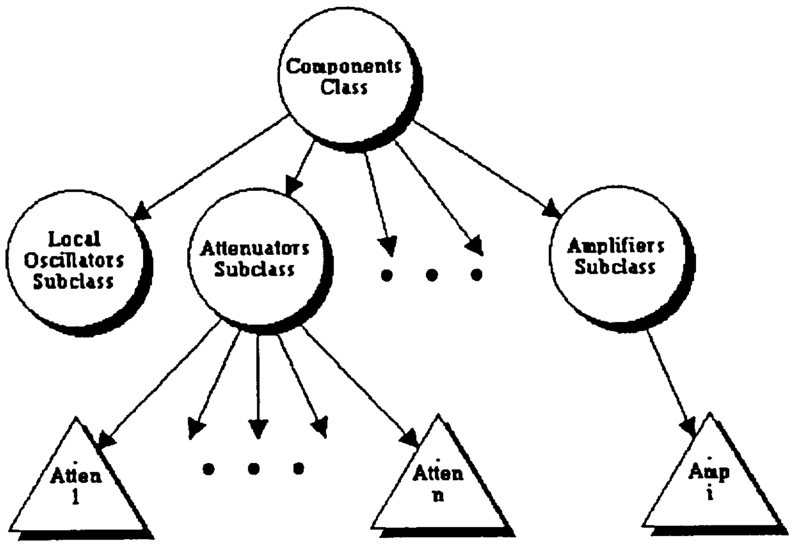

Figure 3.1: Components Class Hierarchy

The root node in Figure 3.1 is a circle indicating a class frame called *Components*. This class was created to represent the commonality between all components in the transponder. It is divided into several subclasses; represented by the second level of class frames. Each of these subclasses describe the function of components in the transponder: amplifiers, attenuators, etc. The components are represented by object frames attached to these subclasses.

3.1.2 Structure of Subsystems Class Hierarchy

Each component is also associated with a subsystem of the transponder, see Figure 3.2. Several object frames are used to represent the collections of components called subsystems. These frames are then attached to a class frame called *Subsystems*.

Finally, the membership of a component to a particular subsystem is represented by attaching its object frame as a subobject of the appropriate subsystem object frame.

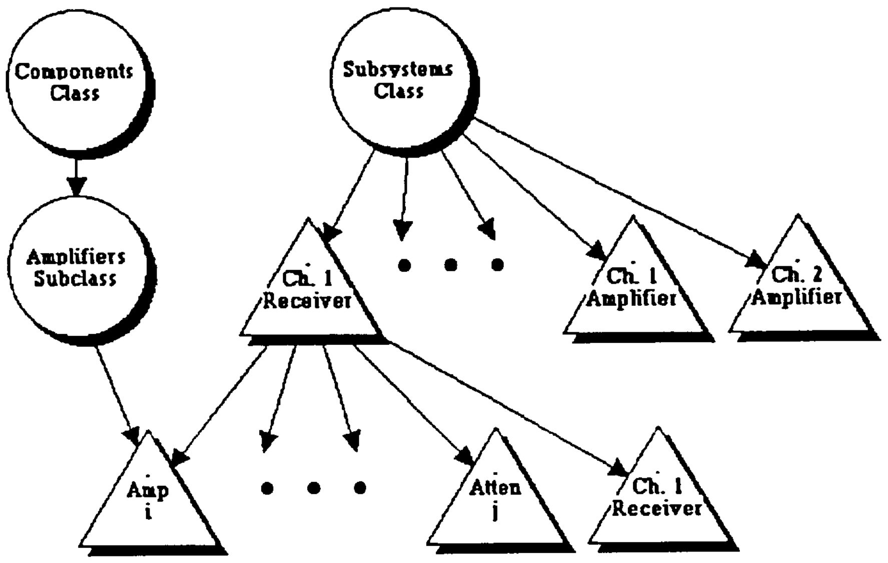

Figure 3.2: Subsystems Class Hierarchy

3.1.3 Structure of Sensors Class Hierarchy

Two types of sensory elements monitor both the response of the transponder and the relayed signal. The first type is signal power level sensors. The other type represents the data stream bit error rate (BER) registers located within the ground terminal systems. The information used for diagnosis is provided by these sensors. These sensors were represented by creating the class structure called *Sensors* for all sensory components shown in Figure 3.3.

This structure is divided into subclasses according to the two types of sensors. Each sensor is then represented by an object attached to the appropriate type subclass.

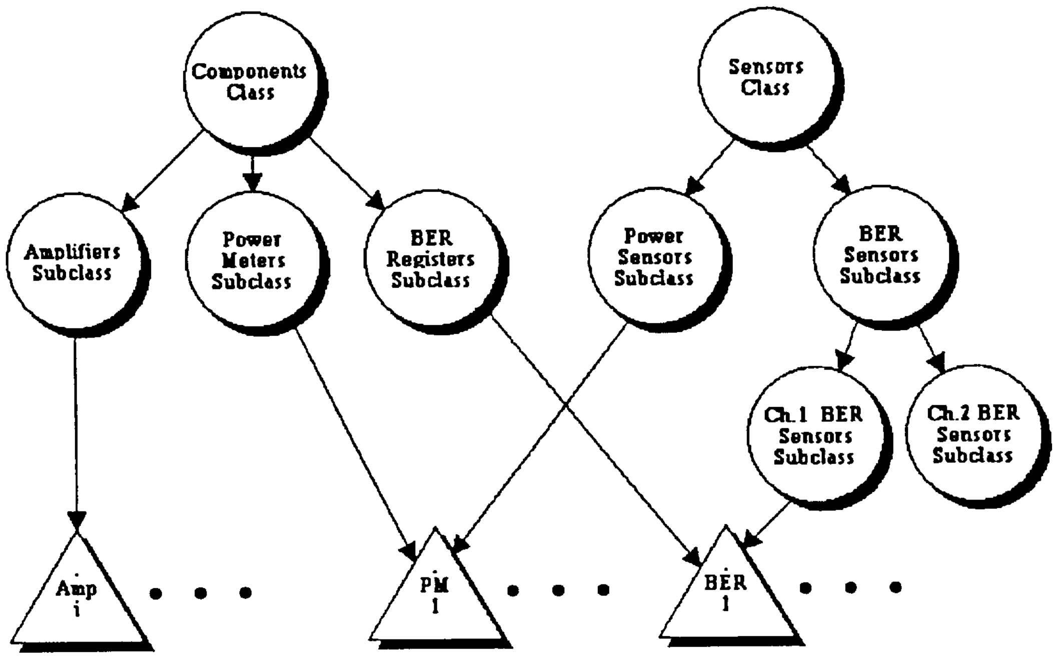

Figure 3.3: Sensors Class Hierarchy

The *BER Sensors* class is also divided into two subclasses according to their channel. This was done to simplify the analysis of frequency dependant fault states. It also demonstrates how class structures can be cascaded to further describe component function and organization.

Like all other transponder components, sensory elements could potentially fail. Therefore, each sensor is also represented in FIDEX as a member of the *Components* world. Each sensory component is represented by an object frame. These frames are linked to their appropriate subclass type in both the components world and the sensors world.

3.1.4 Structure of Fault States Class Hierarchy

The transponder fault states are represented as objects in a class structure called *Fault States*. This class is also divided into several subclasses. Each subclass frame represents the association of fault states to component types such as amplifier faults, attenuator faults, etc. Object frames representing the specific failure modes of the transponder are then attached to the appropriate subclasses. This structure, shown in Figure 3.4, enables FIDEX to reason about both known and abstract faults.

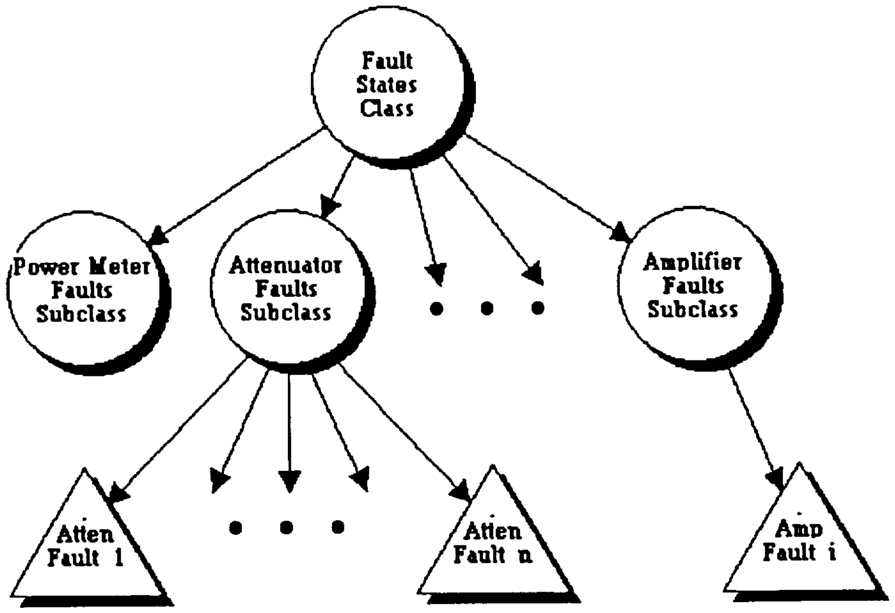
Fault States Class
Power Meter Faults Subclass
Attenuator Faults Subclass
Amplifier Faults Subclass
Atten Fault 1
Atten Fault n
Amp Fault i

Figure 3.4: Fault States Class Hierarchy

This section has discussed the engineering of knowledge about the structure of the transponder and its faut states. The next sections discusses the engineering of knowledge about the diagnosis of fault states within the transponder.

3.2 Modular Approach to Diagnostics

The basic approach was to divide the job of diagnosing faults within the transponder into a series of diagnostic tasks. Each of these tasks was separated into an individual module of the FIDEX system.

The knowledge for the task of monitoring the response of the transponder in order to determine whether it is functioning properly or not was separated into an individual module called the *Fault Detection Module.* Similarly, the knowledge for the task of narrowing the range of suspected components to the smallest possible group was separated into an individual module called the *Fault Isolation and Sensor Validation Module.*

The fault isolation task isolates the probable source of the fault to a subsystem of the transponder. For each of these subsystems, different knowledge is required to investigate the precise nature of a misbehavior and for determining the component causing it. Therefore, this knowledge was separated into a different *Fault Diagnosis Module* for each subsystem of the transponder system. Similarly, the strategies for responding to the diagnosis were also different for each subsystem. Therefore, the fault response task was incorporated into the diagnostic modules for the subsystems.

Each of the above modules are loaded as they are needed in the diagnostic process. In this manner, the FIDEX system functions as a community of experts on the diagnosis of faults in the transponder system.

3.3 Reasoning Techniques

FIDEX reasons with two distinctly different techniques. The first technique, called *absolute reasoning*, is used to establish or reject the existence of predefined fault states. The second technique, called *abstract reasoning*, is used to recover when the diagnostic task cannot reason effectively using the first technique. FIDEX uses the second technique to establish evidence in conceptual fault states.

3.3.1 Absolute Reasoning

In general, *procedural knowledge* which supports rules in absolute terms is *associative knowledge*. This type of knowledge associates conditions with the establishment or rejection of a conclusion. Two types of associative knowledge are used by FIDEX.

The first type is *directly associative*. This knowledge directly associates conditions with conclusions. An example of this type of knowledge might be: *If the data reported by a sensor reading exceeds its tolerance band, then the sensor's reading is "BAD."* The condition of sensor data exceeding its acceptable range is directly associated with establishing a *"BAD"* qualitative description for that reading. Rules which represent this type of knowledge are used to structure the strategies of the diagnostic tasks.

However, the majority of the knowledge used in the task of fault diagnosis is supported by an accumulation of evidence. This type of knowledge is *cumulatively associative*. That is, the accumulation of several conditions is associated with the establishment or rejection of a conclusion. Moreover, each condition may contribute differently to that conclusion. An example of such knowledge might be: *"A LOW signal power level might indicate internal phase lock failure in a local oscillator."* and *"A HIGH bit error rate is might indicate that the local oscillator is out of phase lock."*

Neither conditions can be directly associated to establish or reject the conclusion of an internal phase lock failure. However, each contributes evidence to that conclusion.

When multiple rules contribute evidence toward a conclusion, the system must be able to accumulate this evidence. The FIDEX system has such an ability.

3.3.2 Inexact Reasoning

The FIDEX system uses *inexact reasoning* based on the MYCIN technique for incrementally accumulating evidence during the fault diagnosis task. This is because very few of the known fault states of the transponder system are supported by singular conditions. This section discusses the approach used in FIDEX for implementing the MYCIN technique.

Not shown in Figure 3.4 is an additional node to the fault state hierarchy. The *Fault States* class is attached as a child of another class called *Certainty Analysis.* The certainty analysis class was created to provide the overhead required to perform inexact reasoning. All objects which represent hypotheses requiring certainty analysis are attached to this class. Because the fault state hypotheses require this, attaching their root node to the *Certainty Analysis* class provides them this overhead.

Each fault state object inherits from the *Certainty Analysis* properties for accumulating: *measures of belief, measures of disbelief, accumulated belief MB, MD, AB, AD,* and *CF* quantities that were discussed in section 3.3.2. As the diagnostic process acquires evidence in support or rejection of *Fault State* hypotheses, measures of belief and disbelief are assigned to these properties.

Associated with these properties are algorithms for calculating the confidence factor according to the equations in 3.1 and 3.2. When measures of belief or disbelief are assigned, they are accumulated and an overall certainty is calculated.

3.3.3 Incremental Accumulation of Evidence

FIDEX uses the *incremental accumulation of evidence* technique to establish or reject hypotheses which are supported by multiple rules, Shortliffe [34]. The two equations given in 3.1 accumulate a measure of belief *AB* and disbelief *AD* in a hypothesis *H*. These two measures are then used by Equation 3.2 to establish an overall confidence *CF* in that hypothesis.

$$\begin{aligned} AB(H)_k &- AB(H)_{k-1} + MB(H)_k \cdot [\ 1 - AB(H)_{k-1}\ ] \\ AD(H)_k &- AD(H)_{k-1} + MD(H)_k \cdot [\ 1 - AD(H)_{k-1}\ ] \end{aligned} \qquad (3.1)$$

$$CF(H)_k - [\ \frac{AB(H)_k - AD(H)_k}{1 - \min(AB(H)_k, AD(H)_k)}\ ] \qquad (3.2)$$

Rules which accumulate belief do not assign Boolean values to their associated hypothesis. Instead, they determine a measure of belief *MB* or measure of disbelief *MD* in that conclusion. These measures represent the degree to which the rule has contributed to the establishment or rejection of its hypothesis. The values which are assigned to these measures range between 0 and 1. Values close to 1 represent strong measures while values close to 0 represent weak measures. A value of 1 is generally not assigned; as it results in a Boolean value for *AB* or *AD*.

Consider an arbitrary hypothesis *H* and assume that no evidence has been established toward belief in that conclusion; $K = 0$ and $AB(H)_0 = 0$. Establishing a fact in support of this conclusion might assign a measure of 0.2 to the belief in *H*, for example $MB(H)_1 = 0.2$. According to the first equation in 3.1, the accumulated belief in the hypothesis would then be $AB(H)_1 = 0.2$. The establishment of another fact in support of H might assign a measure of 0.5 to the belief in H, i.e. $MB(H)_2 = 0.5$. The accumulated belief in the hypothesis would then be incremented according to the first equations in 3.1; $AB(H)_2 = 0.6$.

The accumulated measure of disbelief $AD(H)$ is incremented similarly. However, this accumulation would be based on rules which establish measures of disbelief in a hypothesis $MD(H)_k$. This measure indicates evidence in rejection of the hypothesis.

As rules ascribe $MB(H)_k$'s and $MD(H)_k$'s, and accumulated values are calculated, the overall confidence in a conclusion $CF(H)_k$ is calculated. Confidence factors range in value from -1 to 1. A value near -1 signifies little confidence in the hypothesis, or the rejection of the hypothesis. A value near 1 denotes a high level of confidence, or the establishment of the hypothesis. Values in between represent various degrees of confidence, with 0 meaning unknown.

3.3.4 Abstract Reasoning

Discussion to this point has been on the incremental accumulation of evidence toward concrete fault states. The next topic is discuss the application of these techniques for abstract reasoning. In general, knowledge which supports rules in abstract terms is *conceptual knowledge*. This type of knowledge is *indirectly associative knowledge*. It associates conditions to abstract ideas which are indirectly related to the rule being pursued. An example of this type of knowledge might be: *A HIGH bit error rate is typical of a misbehavior in one of the frequency conversion components.*

FIDEX uses this type of reasoning to establish levels of confidence in class level fault categories. That is, it might reach a conclusion of the form: *The observed symptoms are typical of those associated with a failure of the local oscillator.*

During the diagnostic task, FIDEX exhausts its knowledge about the fault states of the system. It is entirely possible that a failure mode might occur for which FIDEX has no knowledge. In that case, it would resort to confidence accumulated in class level fault states as its diagnostic conclusion.

This abstract reasoning ability of FIDEX is implemented as follows. All of the fault state type subclasses defined in section 3.1.4 are attached as subclasses of the class *Certainty Analysis*. Therefore, they inherit information from this class. and permit measures of belief and disbelief to be assigned to the fault state classes. Levels of confidence can then be accumulated at this class, or conceptual, level. Using this technique, FIDEX can piece together information and reach conceptual conclusions about a fault.

3.4 Learning and Adaptive Search Strategy

There are two databases used by FIDEX. One contains information required to initialize parametric values of the system. Each record contains information on nominal readings, error tolerances, and other initial parameters. These values are loaded and stored in the appropriate slots of objects at run time or when FIDEX is initialized. This method of initialization was chosen to facilitate the maintenance of the system.

The second database is used to provide FIDEX a limited learning capability. FIDEX stores the failure history of the transponder system in this database. Each known fault state is represented by a record that contains fields that represent the failure history of that fault state. Following diagnostics, FIDEX increments the history of the identified fault. This record keeping is used to direct the search strategy of future sessions toward the most likely faults.

The search strategy is adaptive in that the priorities by which known fault states are placed on the agenda is based upon the values maintained in the history database. A class level property of all fault states is the integer *INFR_CATEGORY*. The value of this property is retrieved from the database when the diagnostic task is initialized. This property is then assigned to the inference priority of the fault state hypothesis by slot actions. When the diagnostic task establishes a known fault state, the value of its inference category is incremented accordingly. The updated value is then stored in the learning database.

This chapter has discussed the engineering of knowledge about the structure of the transponder system and diagnosing its fault states. The next several chapters discuss the techniques used to represent this knowledge in the knowledgebases of the FIDEX system.

CHAPTER IV
KNOWLEDGE REPRESENTATION

As the previous chapter discussed the engineering of knowledge, this chapter discusses its frame-based representation in the FIDEX system. The kernel of this frame representation of the structure of the transponder is contained in the FIDEX.tkb knowledge base. A complete listing of that knowledge base is included in Appendix A of this report. The sections of this chapter discuss key segments of that knowledge base.

4.1 Representation of Transponder System Components

As discussed in chapter 3, a frame hierarchy was created to provide a clear and efficient representation of all components in the transponder. The root of this structure is a class frame called *COMPONENTS*. This class was created to represent the commonality between all components in the transponder. It is divided into several subclasses; represented by the second level of class frames, as shown in Figure 3.1. The components are represented by object frames attached to these subclasses.

4.1.1 Property Definitions

Code Segment 4.1 shows a series of declarations that define the properties which are used to describe the components of the transponder system. These properties were defined to describe physical characteristics about a component; such as its name, input/output components, etc. Some properties are used by FIDEX to give a component a self awareness. Other properties provide functional information about the components; such as its input and output signal power levels, gain, nominal gain, etc. The properties represent attributes of transponder components as follows.

COMPONENT_IN and *COMPONENT_OUT* are string properties that are used to encode structural information about the transponder components. These property values are initialized for every component to the names of the component objects at their input and output respectively. It is shown in the next chapter how these properties can be used to by an object to obtain information from its neighbors.

Code Segment 4.1: Properties of the *COMPONENTS* Class

```
(@PROPERTY=     COMPONENT_IN              @TYPE=String;)
(@PROPERTY=     COMPONENT_OUT             @TYPE=String;)
(@PROPERTY=     DESCRIPTION               @TYPE=String;)
(@PROPERTY=     FREQUENCY                 @TYPE=Float;)
(@PROPERTY=     FREQUENCY_IN              @TYPE=Float;)
(@PROPERTY=     FREQUENCY_OUT             @TYPE=Float;)
(@PROPERTY=     GAIN                      @TYPE=Float;)
(@PROPERTY=     MODEL_GAIN                @TYPE=Float;)
(@PROPERTY=     MODEL_POWER_IN            @TYPE=Float;)
(@PROPERTY=     MODEL_POWER_OUT           @TYPE=Float;)
(@PROPERTY=     NAME                      @TYPE=String;)
(@PROPERTY=     NASA_ID                   @TYPE=String;)
(@PROPERTY=     NOMINAL_FREQUENCY         @TYPE=Float;)
(@PROPERTY=     NOMINAL_FREQUENCY_IN      @TYPE=Float;)
(@PROPERTY=     NOMINAL_FREQUENCY_OUT     @TYPE=Float;)
(@PROPERTY=     NOMINAL_GAIN              @TYPE=Float;)
(@PROPERTY=     NOMINAL_POWER_IN          @TYPE=Float;)
(@PROPERTY=     NOMINAL_POWER_OUT         @TYPE=Float;)
(@PROPERTY=     POWER_IN                  @TYPE=Float;)
(@PROPERTY=     POWER_OUT                 @TYPE=Float;)
```

Another sting property called *NAME* is used to encode the name of a component object. This allows the object to communicate information about itself through the frame hierarchy. It is also useful for writing generic rules. Such rules operate on information posted in several blackboard frames. This property enables an object to post itself on the blackboard and be operated upon by such rules. This utility is discussed in the next chapter.

NASA_ID and *DESCRIPTION* are string properties which are used to communicate information about a component object through the ToolBook™ interface. *NASA_ID* is initialized to the *tag* which NASA personnel use to reference transponder system components. *DESCRIPTION* is initialized to a functional description of the component.

The remaining properties are used to represent functional attributes of the transponder system components. Floating point properties are used to represent the propagation of the communication signal through a component of the transponder. Particularly, there are two aspects of the signal that are of interest: the signal power level and carrier frequency. The properties *POWER_IN* and *FREQUENCY_IN* are used to represent these attributes of the transponder signal at the input to a component. Similarly, the properties *POWER_OUT* and *FREQUENCY_OUT* are used to represent the signal power level and carrier frequency at the output of a component. The effect that a component has on the signal propagated through it is represented by the floating point properties *GAIN* and *FREQUENCY*. A component's gain is its effect on the power level of the signal passed through it; be that an amplification or attenuation. If a component alters the carrier frequency of the signal, that alteration is represented in the value of the *FREQUENCY* property; be it an upconversion or downconversion.

All components have parametric values for their input/output signal power levels and frequencies. Moreover, their effect on the signal is defined by their design specifications. These nominal values are represented by the properties: *NOMINAL_FREQUENCY, NOMINAL_FREQUENCY_IN, NOMINAL_FREQUENCY_OUT, NOMINAL_GAIN, NOMINAL_POWER_IN,* and *NOMINAL_POWER_OUT*.

Early in the development of the FIDEX system, a direction was taken toward the development of a system that used model-based reasoning. Although this approach was abandoned early on, the properties required for its implementation were left within the *COMPONENTS* world. The reason for this was that it made no sense to destroy this capacity. If NASA would wish to expand this capability, the basic building blocks will exist within the FIDEX frame structure. Specifically, these properties are *MODEL_GAIN*, *MODEL_POWER_IN*, and *MODEL_POWER_OUT*.

4.1.2 Class Definitions

The next definition, Code Segment 4.2, creates a class frame called *COMPONENTS* in the object space of the expert system. It establishes links to several subclasses and defines the properties discussed above as being associated with this class. The ten component subclasses listed represent different types of components in the transponder system.

Several properties are required to represent attributes of specific types of components. These properties do not apply to components in general, but to specific types of transponder components. Code Segment 4.3 lists their definition. Due to the number of properties involved and their distribution between various subclasses, they are discussed with their corresponding subclasses later.

Code Segment 4.2: Definition of the *COMPONENTS* Class

```
(@CLASS=        COMPONENTS
        (@SUBCLASSES=
                AMPLIFIERS
                ATTENUATORS
                LOCAL_OSCILLATORS
                RECEIVERS
                POWER_METERS
                BER_REGISTERS
                SWITCHES
                GaAsFETS
                TWTAS
                MIXERS   )
        (@PROPERTIES=
                COMPONENT_IN
                COMPONENT_OUT
                DESCRIPTION
                FREQUENCY
                FREQUENCY_IN
                FREQUENCY_OUT
                GAIN
                MODEL_GAIN
                MODEL_POWER_IN
                MODEL_POWER_OUT
                NAME
                NASA_ID
                NOMINAL_FREQUENCY
                NOMINAL_FREQUENCY_IN
                NOMINAL_FREQUENCY_OUT
                NOMINAL_GAIN
                NOMINAL_POWER_IN
                NOMINAL_POWER_OUT
                POWER_IN
                POWER_OUT        )        )
```

Code Segment 4.3: Properties of *COMPONENTS* SubClasses

```
(@PROPERTY=     BIAS_CURRENT              @TYPE=Float;)
(@PROPERTY=     BIAS_VOLTAGE              @TYPE=Float;)
(@PROPERTY=     COMPONENT_IN_2            @TYPE=String;)
(@PROPERTY=     COMPONENT_OUT_2           @TYPE=String;)
(@PROPERTY=     CONFIG                    @TYPE=String;)
(@PROPERTY=     DRAIN_VOLTAGE             @TYPE=Float;)
(@PROPERTY=     FREQUENCY_2               @TYPE=Float;)
(@PROPERTY=     FREQUENCY_IN_2            @TYPE=Float;)
(@PROPERTY=     FREQUENCY_OUT_2           @TYPE=Float;)
(@PROPERTY=     GAIN_2                    @TYPE=Float;)
(@PROPERTY=     GATE_VOLTAGE              @TYPE=Float;)
(@PROPERTY=     LO_INPUT_FREQUENCY        @TYPE=Float;)
(@PROPERTY=     LO_INPUT_POWER            @TYPE=Float;)
(@PROPERTY=     LO_UNIT                   @TYPE=String;)
(@PROPERTY=     MODEL_GAIN_2              @TYPE=Float;)
(@PROPERTY=     MODEL_POWER_IN_2          @TYPE=Float;)
(@PROPERTY=     MODEL_POWER_OUT_2         @TYPE=Float;)
(@PROPERTY=     MODEL_SETTING             @TYPE=Float;)
(@PROPERTY=     NOMINAL_BIAS_CURRENT      @TYPE=Float;)
(@PROPERTY=     NOMINAL_BIAS_VOLTAGE      @TYPE=Float;)
(@PROPERTY=     NOMINAL_DRAIN_VOLTAGE     @TYPE=Float;)
(@PROPERTY=     NOMINAL_FREQUENCY_2       @TYPE=Float;)
(@PROPERTY=     NOMINAL_FREQUENCY_IN_2              @TYPE=Float;)
(@PROPERTY=     NOMINAL_FREQUENCY_OUT_2             @TYPE=Float;)
(@PROPERTY=     NOMINAL_GAIN_2            @TYPE=Float;)
(@PROPERTY=     NOMINAL_GATE_VOLTAGE      @TYPE=Float;)
(@PROPERTY=     NOMINAL_LO_INPUT_FREQUENCY          @TYPE=Float;)
(@PROPERTY=     NOMINAL_LO_INPUT_POWER              @TYPE=Float;)
(@PROPERTY=     NOMINAL_POWER_IN_2        @TYPE=Float;)
(@PROPERTY=     NOMINAL_POWER_OUT_2       @TYPE=Float;)
(@PROPERTY=     NOMINAL_SETTING           @TYPE=Float;)
(@PROPERTY=     POWER_IN_2                @TYPE=Float;)
(@PROPERTY=     POWER_OUT_2               @TYPE=Float;)
(@PROPERTY=     SETTING                   @TYPE=Float;)
(@PROPERTY=     SETTING_ERROR             @TYPE=Float;)
```

Each subclass defined in Code Segments 4.4a and 4.4b represent the subclass nodes introduced in Figure 3.1 in section 3.1.1. Again one class is created for each basic type of category of component in the transponder system. The names are descriptive but are explained for clarity.

Code Segment 4.4a: Definition of the *COMPONENTS* SubClasses

```
(@CLASS=        AMPLIFIERS
        (@PROPERTIES=
                BIAS_CURRENT
                BIAS_VOLTAGE
                DRAIN_VOLTAGE
                GATE_VOLTAGE
                NOMINAL_BIAS_CURRENT
                NOMINAL_BIAS_VOLTAGE
                NOMINAL_DRAIN_VOLTAGE
                NOMINAL_GATE_VOLTAGE   )        )

(@CLASS=        ATTENUATORS
        (@PROPERTIES=
                MODEL_SETTING
                NOMINAL_SETTING
                SETTING
                SETTING_ERROR   )       )

(@CLASS=        BER_REGISTERS   )

(@CLASS=        GaAsFETS
        (@PROPERTIES=
                DRAIN_VOLTAGE
                GATE_VOLTAGE
                NOMINAL_DRAIN_VOLTAGE
                NOMINAL_GATE_VOLTAGE   )        )

(@CLASS=        LOCAL_OSCILLATORS
        (@PROPERTIES=
                COMPONENT_OUT_2
                FREQUENCY_OUT_2
                NOMINAL_FREQUENCY_OUT_2
                NOMINAL_POWER_OUT_2
                POWER_OUT_2     )       )
```

The subclass called *AMPLIFIERS* is used to classify objects that represent components that amplify the power level of the signal inside the transponder system. Associated with this class are 8 floating point properties. The first two properties listed in definition of *AMPLIFIERS* in Code Segment 4.4a are *BIAS_CURRENT* and *BIAS_VOLTAGE.*

These properties represent the bias current and bias voltage supplied to an amplifier component by its power supply. The second two properties, *DRAIN_VOLTAGE* and *GATE_VOLTAGE* are used to represent the voltage levels on an amplifier component's drain and gate respectively. The design parameters specify nominal values for these four quantities. The remaining properties, prefixed by *NOMINAL_* are used to represent the respective parametric values.

The subclass called *ATTENUATORS* is used to classify objects that represent components that attenuate the power level of the signal inside the transponder system. Again, several properties are unique to these objects. Attenuator objects have only one unique attribute. This is that they have a *SETTING.* The levels of attenuation for these components are set either manually or by the Network Control Computer (NCC). This value is stored in the floating point property called *SETTING.* The parametric value for an attenuator's setting is represented by the *NOMINAL_SETTING* property. The difference between an attenuator's nominal setting and its actual setting is represented by the property called *SETTING_ERROR.* And finally, a remanent of the model-based overhead is provided for an attenuators *MODEL_SETTING.*

The third definition in Code Segment 4.4a creates a class called *BER_REGISTERS.* This class is used to represent the Bit Error Rate Register objects as components of the transponder system. The roles of sensors as both sensor elements and transponder system components were discussed in section 3.1.3 of chapter 3.

The fourth class definition creates a class for the GaAs FET Amplifier. This shares four properties with the *AMPLIFIERS* class. However, the *GaAsFETS* class is separated from the amplifiers because the bias current and bias voltage properties have no meaning for Gallium Arsenide Field Effect Transistor (GaAs FET) amplifiers. The properties *DRAIN_* and *GATE_VOLTAGE* and their associated *NOMINAL_* values for the GaAs FETs are the same as for the amplifiers.

The final class definition in Code Segment 4.4a creates a classification for the objects that represent local oscillators (LO). The *LOCAL_OSCILLATORS* class has five unique properties. However, all of these as extensions of concepts which have already been discussed. That is, the local oscillators in the ACTS transponder system are multiple output devices. Each unit has one output per channel through the transponder. In the current phase of development, only two channels are operating. This requires that an additional output port be represented in the properties of the objects that represent Los. Therefore, the properties that represent the functional and relational attributes of a component's output port were duplicated and suffixed by *_2*.

Code Segment 4.4b continues the definition of the classes which organize the *COMPONENTS* world hierarchy. The first definition creates a class for the objects that represent the two multiplexers, or signal mixers. These units have an additional input for a signal from a local oscillator. Therefore, the properties for component input parameters were duplicated and prefixed with *LO_* to represent this additional signal. The property *LO_UNIT* corresponds to the *COMPONENT_IN* property discussed earlier, except that this property represents the name of the local oscillator associated with the LO input port.

The second definition in Code Segment 4.4b creates a class called *PWR_METERS*. This class is used to represent the power meter objects as components of the transponder system.

The third definition creates a class called *RECEIVERS* for the objects that represent the two receiver components in the transponder. As for the *MIXERS*, these components also have a LO input port. Therefore, this class has the same properties as the mixers class and the corresponding properties represent the same quantities.

Code Segment 4.4b: Definition of the *COMPONENTS* SubClasses

```
(@CLASS=          MIXERS
         (@PROPERTIES=
                  LO_INPUT_FREQUENCY
                  LO_INPUT_POWER
                  LO_UNIT
                  NOMINAL_LO_INPUT_FREQUENCY
                  NOMINAL_LO_INPUT_POWER )          )

(@CLASS=          POWER_METERS    )

(@CLASS=          RECEIVERS
         (@PROPERTIES=
                  LO_INPUT_FREQUENCY
                  LO_INPUT_POWER
                  LO_UNIT
                  NOMINAL_LO_INPUT_FREQUENCY
                  NOMINAL_LO_INPUT_POWER )          )

(@CLASS=          SWITCHES
         (@PROPERTIES=
                  COMPONENT_IN_2
                  COMPONENT_OUT_2
                  FREQUENCY_2
                  FREQUENCY_IN_2
                  FREQUENCY_OUT_2
                  GAIN_2
                  MODEL_GAIN_2
                  MODEL_POWER_IN_2
                  MODEL_POWER_OUT_2
                  NOMINAL_FREQUENCY_2
                  NOMINAL_FREQUENCY_IN_2
                  NOMINAL_FREQUENCY_OUT_2
                  NOMINAL_GAIN_2
                  NOMINAL_POWER_IN_2
                  NOMINAL_POWER_OUT_2
                  POWER_IN_2
                  POWER_OUT_2      )         )

(@CLASS=          TWTAS   )
```

The fourth class definition in Code Segment 4.4b creates a class for the matrix switch. This component is a multiple input/output device having one input and one output per channel in the transponder system. Since there are two channels, one additional input and one additional output port needed to be represented in the properties of the *SWITCHES* class. Again, these are simply extensions of concepts discussed for the general case of *COMPONENTS*, and they are suffixed by *_2*.

The final class definition for the subclasses of the *COMPONENTS* world is *TWTAS*. This class is for the Traveling Wave Tube Amplifier (TWTA). There are no unique properties associated with this component.

4.1.3 Object Definitions

The next step is to create objects to represent the various components of the transponder system and link them to their respective component type subclasses. These definitions are given in Code Segments 4.5a and 4.5b. Each object corresponds to one of the transponder system components that were introduced in section 1.1.2 of chapter 1. They are listed in Table 1.1.

The first definition in Code Segment 4.5a creates an object to represent the Gallium-Arsenide Field Effect Transistor (GaAs FET) amplifier. This unit is located at the output of the channel 1 signal path, see Figure 1.2. This object, called *GAASFET*, is linked as a child of the *GaAsFETS* class. Therefore, it inherits all the associated properties that were discussed in the previous section.

The next two definitions create objects to represent the High Power Amplifier Input Power Control (HPAPC) amplifiers labeled as *E* and *F* in Figure 1.2. *HPAPC_AMP_1* represents the power control amplifier in the channel 1 output signal path and *HPAPC_AMP_2* represents its counterpart in channel 2. Since both these objects represent amplifiers in the transponder, they are both attached to the *AMPLIFIERS* subclass of the *COMPONENTS* hierarchy.

The remaining HPAPC components are the attenuators labeled as *K, L, M,* and *N* in Figure 1.2. These components are represented by the objects created as *HPAPC_ATTN_1* through *HPAPC_ATTN_4*. The fourth through seventh definitions in Code Segment 4.5a create these objects and attach them to the *ATTENUATORS* class.

Code Segment 4.5a: Objects of the *COMPONENTS* Class

```
(@OBJECT=        GAASFET
         (@CLASSES=       GaAsFETS           )        )

(@OBJECT=        HPAPC_AMP_1
         (@CLASSES=       AMPLIFIERS         )        )

(@OBJECT=        HPAPC_AMP_2
         (@CLASSES=       AMPLIFIERS         )        )

(@OBJECT=        HPAPC_ATTN_1
         (@CLASSES=       ATTENUATORS        )        )

(@OBJECT=        HPAPC_ATTN_2
         (@CLASSES=       ATTENUATORS        )        )

(@OBJECT=        HPAPC_ATTN_3
         (@CLASSES=       ATTENUATORS        )        )

(@OBJECT=        HPAPC_ATTN_4
         (@CLASSES=       ATTENUATORS        )        )

(@OBJECT=        IFPC_AMP_1
         (@CLASSES=       AMPLIFIERS         )        )

(@OBJECT=        IFPC_AMP_2
         (@CLASSES=       AMPLIFIERS         )        )

(@OBJECT=        IFPC_AMP_3
         (@CLASSES=       AMPLIFIERS         )        )

(@OBJECT=        IFPC_AMP_4
         (@CLASSES=       AMPLIFIERS         )        )

(@OBJECT=        IFPC_ATTN_1
         (@CLASSES=       ATTENUATORS        )        )

(@OBJECT=        IFPC_ATTN_2
         (@CLASSES=       ATTENUATORS        )        )

(@OBJECT=        IFPC_ATTN_3
         (@CLASSES=       ATTENUATORS        )        )

(@OBJECT=        IFPC_ATTN_4
         (@CLASSES=       ATTENUATORS        )        )
```

The remaining definitions in Code Segment 4.5a create objects to represent the Intermediate Frequency Power Control (IFPC) components within the transponder system. These were indicated in Figure 1.2 by labels *A* through *J*. The first four, *IFPC_AMP_1* through *_4*, represent the IFPC amplifiers. They are therefore linked as children of the *AMPLIFIERS* subclass. The remaining four, *IFPC_ATTN_1* through *_4*, represent the IFPC attenuators. They are therefore linked as children of the *ATTENUATORS* subclass.

Code Segment 4.5b continues the definition of objects that represent the components of the transponder system. The first two definitions create a new property called *CONFIG* and an object called *MSWITCH* that represents the matrix switch component. It is attached as a child of the *SWITCHES* subclass in the *COMPONENTS* world hierarchy. This object has a unique property called *CONFIG* that is used to represent the multiple channel handling of the matrix switch. The specifics of this property are discussed in the next chapter.

Code Segment 4.5b: Objects of the *COMPONENTS* Class

```
(@PROPERTY=      CONFIG              @TYPE=String;)

(@OBJECT=        MSWITCH
        (@CLASSES=      SWITCHES          )
        (@PROPERTIES=   CONFIG  )         )

(@OBJECT=        MULT_1
        (@CLASSES=      MIXERS  )         )

(@OBJECT=        MULT_2
        (@CLASSES=      MIXERS  )         )

(@OBJECT=        RCVR_1
        (@CLASSES=      RECEIVERS         )         )

(@OBJECT=        RCVR_2
        (@CLASSES=      RECEIVERS         )         )

(@OBJECT=        RCVR_LO
        (@CLASSES=      LOCAL_OSCILLATORS         )         )

(@OBJECT=        TWTA
        (@CLASSES=      TWTAS   )         )

(@OBJECT=        UPX_LO
        (@CLASSES=      LOCAL_OSCILLATORS         )         )
```

The next two definitions create objects, *MULT_1* and *MULT_2*, to represent the up-converter multiplexers. These components were indicated in Figure 1.2 and Table 1.1

as *CH1MIX* and *CH2MIX*. Both these objects are attached to the *MIXERS* subclass of the *COMPONENTS* hierarchy. They therefore inherit all properties that were discussed in the previous sections.

The definitions in Code Segment 4.5b continue with the definition of two objects to represent the receiver units at the inputs to the transponder system. The object named *RCVR_1* represents the Channel 1 Receiver Unit that was labeled as *CH1RCVR* in Figure 1.2 and Table 1.1. The object named *RCVR_2* represents the Channel 2 Receiver Unit that was labeled as *CH2RCVR*. Both these objects are attached as children of the *RECEIVERS* class discussed in the previous section.

Objects that represent the local oscillator units in the transponder are created and attached to the *LOCAL_OSCILLATORS* class. The object called *RCVR_LO* represents the Receiver Unit Local Oscillator that drives the receiver units. The object called *UPX_LO* represents the Up-converter Mixer Local Oscillator that drives the up-converter mixers.

Finally, the Traveling Wave Tube Amplifier (TWTA) is represented by the creation of an object called *TWTA* attached to the *TWTAS* subclass. This completes the definition of the Class/SubClass/Object hierarchy that was introduced in section 3.1.1 of chapter 3. The next section of this chapter discusses the representation of the Subsystems Class that was introduced in section 3.1.2 of chapter 3.

4.2 Representation of Transponder SubSystems

It was shown in Figure 3.2 how each component of the transponder is associated with a subsystem of the transponder. Several object frames are used to represent the collections of components called subsystems. These frames are then organized by attaching them to a class frame for all subsystems in the transponder. Finally, the membership of a component to a particular subsystem is represented by attaching its object frame as a subobject of the appropriate subsystem object frame. The following code segments define this hierarchy.

4.2.1 Property Definitions

Code Segment 4.6 shows a series of declarations that define the properties which are to be used to describe the subsystems of the transponder system. The first property is called *DIAGNOSTIC_MODULE*. Recall from section 1.3 of the introduction that the idea of a transponder subsystem was developed for the isolation of a fault. Once a fault is isolated to a subsystem of the transponder, the next step is to load a diagnostic module to perform the task of fault diagnosis on that subsystem. This Boolean property is initialized to load the diagnostic knowledge base that corresponds to a particular subsystem. The details of this are discussed in the following chapter.

Code Segment 4.6: Properties of the *SUBSYSTEMS* Class

```
(@PROPERTY=     DIAGNOSTIC_MODULE      @TYPE=Boolean;)
(@PROPERTY=     ISOLATED               @TYPE=Boolean;)
(@PROPERTY=     LEVEL_IN               @TYPE=String;)
(@PROPERTY=     LEVEL_OUT              @TYPE=String;)
(@PROPERTY=     READING_IN             @TYPE=String;)
(@PROPERTY=     READING_OUT            @TYPE=String;)
(@PROPERTY=     SENSOR_IN              @TYPE=String;)
(@PROPERTY=     SENSOR_OUT             @TYPE=String;)
(@PROPERTY=     SUBSYSTEM_IN           @TYPE=String;)
(@PROPERTY=     SUBSYSTEM_OUT          @TYPE=String;)
```

The Boolean property called *ISOLATED* is used to flag a subsystem as being the probable source of a detected fault. The rule knowledge used in isolating a fault is discussed in chapter 7. The actions of these rules set this flag to indicate a subsystem has been isolated.

The next four definitions in Code Segment 4.6 are for string properties to describe the signal power levels at the input and output of a subsystem. The *READING_IN* and *READING_OUT* properties are set to the qualitative descriptions, *"GOOD"* or *"BAD,"* ascribed to sensor readings during the fault detection. The *LEVEL_IN* and *LEVEL_OUT* properties are set to the qualitative descriptions, *"HIGH," "LOW", "ZERO,"* or *"OK,"* that are also ascribed to signal power levels during fault detection.

The remaining definitions create properties to describe structural information about the subsystems of the transponder. The string properties *SENSOR_IN* and *SENSOR_OUT* are initialized to the name of the sensor object at a subsystem's input and output respectively. Similarly, the string properties *SUBSYSTEM_IN* and *SUBSYSTEM_OUT* are initialized to the name of the subsystem object at a subsystem's input and output respectively.

4.2.2 Class Definition

The next definition, Code Segment 4.7, creates a class frame called *SUBSYSTEMS* in the object space of the expert system. The properties discussed in the previous section are assigned to this class and are inherited by all attached object frames.

Code Segment 4.7: Definition of the *SUBSYSTEMS* Class

```
(@CLASS=        SUBSYSTEMS
          (@PROPERTIES=
                DIAGNOSTIC_MODULE
                ISOLATED
                LEVEL_IN
                LEVEL_OUT
                NAME
                READING_IN
                READING_OUT
                SENSOR_IN
                SENSOR_OUT
                SUBSYSTEM_IN
                SUBSYSTEM_OUT   )        )
```

4.2.3 Object Definitions

There are seven subsystems in the transponder system. Each is represented by an object attached as a child of the *SUBSYSTEMS* class. Code Segment 4.8 lists the definition for six of these.

First, an object is created to represent the Channel 1 Amplifier Subsystem. Its object name is *CH1AMP*. The GaAs FET amplifier is the only transponder component in this subsystem. The object that represents this component, *GAASFET*, is attached as a subobject of the *CH1AMP* object.

Second, the Channel 1 Receiver Subsystem is represented by creating an object called *CH1RCVR* and attaching it to the *SUBSYSTEMS* class. There are four components in this subsystem. The objects which represent these components, *IFPC_AMP_1*, *IFPC_ATTN_1*, *RCVR_1*, and *RCVR_LO*, are attached as subobjects of the *CH1RCVR* subsystem.

The object that represents the Channel 1 Up-converter Subsystem is defined in Code Segment 4.8 as *CH1UPX*. Its subobjects are the *HPAPC_AMP_1*, *HPAPC_ATTN_1*, *HPAPC_ATTN_2*, *MULT_1*, and *UPX_LO*.

Code Segment 4.8: Objects of the *SUBSYSTEMS* Class

```
(@OBJECT=          CH1AMP
        (@CLASSES=         SUBSYSTEMS          )
        (@SUBOBJECTS=      GAASFET )           )

(@OBJECT=          CH1RCVR
        (@CLASSES=         SUBSYSTEMS          )
        (@SUBOBJECTS=      IFPC_AMP_1
                           IFPC_ATTN_1
                           RCVR_1
                           RCVR_LO )           )

(@OBJECT=          CH1UPX
        (@CLASSES=         SUBSYSTEMS          )
        (@SUBOBJECTS=      HPAPC_AMP_1
                           HPAPC_ATTN_1
                           HPAPC_ATTN_2
                           MULT_1
                           UPX_LO  )           )

(@OBJECT=          CH2AMP
        (@CLASSES=         SUBSYSTEMS          )
        (@SUBOBJECTS=      TWTA     )          )

(@OBJECT=          CH2RCVR
        (@CLASSES=         SUBSYSTEMS          )
        (@SUBOBJECTS=      IFPC_AMP_2
                           IFPC_ATTN_2
                           RCVR_2
                           RCVR_LO )           )

(@OBJECT=          CH2UPX
        (@CLASSES=         SUBSYSTEMS          )
        (@SUBOBJECTS=      HPAPC_AMP_2
                           HPAPC_ATTN_3
                           HPAPC_ATTN_4
                           MULT_2
                           UPX_LO  )           )
```

Fourth, an object is created to represent the Channel 2 Amplifier Subsystem. Its object name is *CH2AMP*. The Traveling Wave Tube amplifier is the only transponder component in this subsystem. The object that represents this component, *TWTA*, is attached as a subobject of the *CH2AMP* object.

Next, the Channel 2 Receiver Subsystem is represented by creating an object called *CH2RCVR* and attaching it to the *SUBSYSTEMS* class. There are four components in this subsystem. The objects which represent these components, *IFPC_AMP_2*, *IFPC_ATTN_2*, *RCVR_2*, and *RCVR_LO*, are attached as subobjects of the *CH2RCVR* subsystem.

Finally, the object that represents the Channel 2 Up-converter Subsystem is defined in Code Segment 4.8 as *CH2UPX.* Its subobjects are the *HPAPC_AMP_2, HPAPC_ATTN_3, HPAPC_ATTN_4, MULT_2,* and *UPX_LO.*

The remaining subsystem is the Matrix Switch Subsystem. There are five components associated with this subsystem. These are represented by the *IFPC_AMP_3, IFPC_AMP_4, IFPC_ATTN_3, IFPC_ATTN_4,* and *MSWITCH* component objects. However, the definition of the *SUBSYSTEMS* objects that represent this group of components differs from the previous.

Recall from section 1.2.2 of the introduction that there are multiple permutations through the matrix switch. The interconnectivity through the matrix switch was detailed in Table 1.2 in chapter 1. To represent each of these signal paths, four objects are required. The definitions for these are given in Code Segments 4.9a and 4.9b.

These objects are not attached as children of the *SUBSYSTEMS* class. Rather, they are left independent. During run time, two of these objects are dynamically linked to the *SUBSYSTEMS* class. This dynamic attachment is based upon the configuration of the matrix switch at the time a diagnostic session begins. The dynamics of this configuration is discussed in the next chapter.

The objects defined in Code Segment 4.9a represent the signal paths through the matrix switch subsystem in its primary configuration. This configuration is: Channel 1 Input routed to Channel 1 Output and Channel 2 Input routed to Channel 2 Output. In later discussions, this configuration is referred to as matrix switch configuration *A.*

Code Segment 4.9a: Dynamic Objects of the *SUBSYSTEMS* Class

```
(@OBJECT=          MSWITCH_CH11
        (@SUBOBJECTS=   MSWITCH
                        IFPC_AMP_3
                        IFPC_ATTN_3        )
        (@PROPERTIES=
                DIAGNOSTIC_MODULE
                ISOLATED
                LEVEL_IN
                LEVEL_OUT
                NAME
                READING_IN
                READING_OUT
                SENSOR_IN
                SENSOR_OUT
                SUBSYSTEM_IN
                SUBSYSTEM_OUT   )          )

(@OBJECT=          MSWITCH_CH22
        (@SUBOBJECTS=   MSWITCH
                        IFPC_AMP_4
                        IFPC_ATTN_4        )
        (@PROPERTIES=
                DIAGNOSTIC_MODULE
                ISOLATED
                LEVEL_IN
                LEVEL_OUT
                NAME
                READING_IN
                READING_OUT
                SENSOR_IN
                SENSOR_OUT
                SUBSYSTEM_IN
                SUBSYSTEM_OUT   )          )
```

The objects defined in Code Segment 4.9b represent the signal paths through the matrix switch subsystem in its secondary configuration. This configuration is: Channel 1 Input routed to Channel 2 Output and Channel 2 Input routed to Channel 1 Output. In later discussions, this configuration is referred to as matrix switch configuration *B*.

Code Segment 4.9b: Dynamic Objects of the *SUBSYSTEMS* Class

```
(@OBJECT=           MSWITCH_CH12
          (@SUBOBJECTS=    MSWITCH
                           IFPC_AMP_4
                           IFPC_ATTN_4        )
          (@PROPERTIES=
                     DIAGNOSTIC_MODULE
                     ISOLATED
                     LEVEL_IN
                     LEVEL_OUT
                     NAME
                     READING_IN
                     READING_OUT
                     SENSOR_IN
                     SENSOR_OUT
                     SUBSYSTEM_IN
                     SUBSYSTEM_OUT   )        )

(@OBJECT=           MSWITCH_CH22
          (@SUBOBJECTS=    MSWITCH
                           IFPC_AMP_3
                           IFPC_ATTN_3        )
          (@PROPERTIES=
                     DIAGNOSTIC_MODULE
                     ISOLATED
                     LEVEL_IN
                     LEVEL_OUT
                     NAME
                     READING_IN
                     READING_OUT
                     SENSOR_IN
                     SENSOR_OUT
                     SUBSYSTEM_IN
                     SUBSYSTEM_OUT   )        )
```

As these frames represent components of the transponder, they are attached to the *COMPONENTS* class structure as well. This linking of component object frames to the components world can be interpreted as an *Is-A Link*. Links to the subsystems world represents *Part-Of Links*. That is, the IFPC Amplifier *Is A*n amplifier and is *Part Of* the Channel 1 Receiver system.

This approach not only aids the diagnostic tasks, but provides an efficient coding approach. Through multiple inheritance, each subsystem component acquires information from two parents. One provides information on performance while the other on structure.

4.3 Representation of Sensory Components

Two types of sensory elements monitor both the response of the transponder and the relayed signal. The first type is signal power level sensors. The other type represents the data stream bit error rate (BER) registers located within the ground terminal systems. The information used for diagnosis is provided by these sensors.

This structure is divided into subclasses according to the two types of sensors. Each sensor is then represented by an object attached to the appropriate type subclass. The following code segments create this structure in the object space of the expert system.

4.3.1 Property Definitions

Properties are defined in Code Segment 4.10 to describe the *DATA* reported by a sensor, its *NOMINAL* value, the corresponding *ERROR*, and the *TOLERANCE* band of acceptable error magnitudes. A string property called *READING* is used for the qualitative descriptions which were introduced in section 1.3.

The string property *LEVEL* is used for a qualitative description of the signal power level reported by a sensor. This property is very important to the modules which perform diagnostics on the individual subsystems of the transponder. Its utility is discussed in great detail in subsequent chapters. However, the floating point property *ZERO_LEVEL* is associated with this qualitative description. This property value is initialized to the sensor reading below which the sensor can be assumed to be reporting a *"Zero"* value.

Code Segment 4.10: Properties of the *SENSORS* Class

```
(@PROPERTY=     DATA           @TYPE=Float;)
(@PROPERTY=     ERROR          @TYPE=Float;)
(@PROPERTY=     EVALUATED      @TYPE=Boolean;)
(@PROPERTY=     LEVEL          @TYPE=String;)
(@PROPERTY=     NOMINAL        @TYPE=Float;)
(@PROPERTY=     READING        @TYPE=String;)
(@PROPERTY=     RTN_LEVEL      @TYPE=Boolean;)
(@PROPERTY=     RTN_NOMINAL    @TYPE=Boolean;)
(@PROPERTY=     RTN_READING    @TYPE=Boolean;)
(@PROPERTY=     TOLERANCE      @TYPE=Float;)
(@PROPERTY=     TYPE           @TYPE=String;)
(@PROPERTY=     ZERO_LEVEL     @TYPE=Float;)
```

Once a sensor has been evaluated, a Boolean property called *EVALUATED* is set to *TRUE*. This property is used to poll a sensor to determine if its reported sensor data has been evaluated. A value of *TRUE* implies that the current descriptions of *READING* and *LEVEL* reflect the current reported *DATA* value.

The remaining properties for *SENSORS* class are those required by the ToolBook™ Graphical User Interface (GUI). The string property *TYPE* is used to communicate the type of sensor, *"BER"* or *"PM,"* which is communicating information to the GUI. The other properties, prefixed with *RTN_*, are used to initiate communication of sensor *LEVEL* and *READING* descriptions as well as *NOMINAL* sensor data values through the GUI.

4.3.2 Class Definitions

The *SENSORS* class hierarchy was introduced in chapter 3. The definitions which create the structure shown in Figure 3.3 are given in Code Segment 4.11.

Code Segment 4.11: *SENSORS* Class Hierarchy

```
(@CLASS=        SENSORS
        (@SUBCLASSES=
                PWR_SENSORS
                BER_SENSORS       )
        (@PROPERTIES=
                DATA
                ERROR
                EVALUATED
                LEVEL
                NAME
                NOMINAL
                READING
                RTN_LEVEL
                RTN_NOMINAL
                RTN_READING
                TOLERANCE
                TYPE
                ZERO_LEVEL        )        )

(@CLASS=        PWR_SENSORS       )

(@CLASS=        BER_SENSORS
        (@SUBCLASSES=
                CH1_BERs
                CH2_BERs          )        )

(@CLASS=        CH1_BERs          )

(@CLASS=        CH2_BERs          )

(@CLASS=        BAD_SENSORS
        (@PROPERTIES=   RTN_READING        )        )
```

The first definition creates the *SENSORS* class in the object space of the FIDEX system. This definition attaches two subclasses to the *SENSORS* frame. The class called *PWR_SENSORS* is used to classify objects which represent signal power level sensors. The second class, called *BER_SENSORS* is used to classify objects which represent data stream bit error rate registers. The remainder of the definition for the *SENSORS* class attaches the properties discussed in the previous section to this hierarchy.

Notice that the string property *NAME* is also associated with the *SENSORS* class. This property was defined with the *COMPONENTS* class and was therefore not redefined in Code Segment 4.10. This property is used in the same context with sensors as it was for components. It allows sensor objects to post their names in blackboard property values and communicate with other objects and generic rules.

The next two definitions in Code Segment 4.11 create the classes *PWR_SENSORS* and *BER_SENSORS* in the object space of the FIDEX system. The *BER SENSORS* class is also divided into two subclasses according to their channel; *CH1_BERs* and *CH2_BERs*. This was done to simplify the analysis of frequency dependant fault states. It also demonstrates how class structures can be cascaded to further describe component function and organization. The fourth and fifth definitions create these classes.

The final definition is Code Segment 4.11 creates a class called *BAD_SENSORS*. This class is used as a list of sensors which report "*BAD*" sensor readings. This class is not a *SENSORS* subclass, but it is associated with the *SENSORS* world. The only property used in connection with this class is the *RTN_READING* property. This is required to return a list of the sensors evaluated as having "*BAD*" readings to the GUI.

4.3.3 Object Definitions

Each sensory component is represented by an object frame. These frames are linked to their appropriate type subclass in both the components world, and the sensors world. The definitions which create objects to represent sensory components are given in Code Segments 4.12a and 4.12b. Code Segment 4.12a lists the definitions for the BER registers. *BER_1, BER_2,* and *BER_3* are associated with the channel 1 user data stream. They are therefore attached to the *SENSORS* hierarchy as *CH1_BERs*. *BER_4, BER_5,* and *BER_6* are associated with the channel 2 user data stream. They are therefore attached to the *SENSORS* hierarchy as *CH2_BERs*.

Like all other transponder components, sensory elements could potentially fail. Therefore, each BER sensor is also represented FIDEX as a member of the component world; belonging to the class of *BER_REGISTERS* that was discussed in section 2.3 and section 4.1.

Code Segment 4.12a: *BER_SENSOR* Objects

```
(@OBJECT=        BER_1
         (@CLASSES=
                 CH1_BERs
                 BER_REGISTERS    )         )

(@OBJECT=        BER_2
         (@CLASSES=
                 CH1_BERs
                 BER_REGISTERS    )         )

(@OBJECT=        BER_3
         (@CLASSES=
                 CH1_BERs
                 BER_REGISTERS    )         )

(@OBJECT=        BER_4
         (@CLASSES=
                 CH2_BERs
                 BER_REGISTERS    )         )

(@OBJECT=        BER_5
         (@CLASSES=
                 CH2_BERs
                 BER_REGISTERS    )         )

(@OBJECT=        BER_6
         (@CLASSES=
                 CH2_BERs
                 BER_REGISTERS    )         )
```

Code Segment 4.12b lists the definitions for the signal power level sensors. These eight sensors were listed in Table 1.1. The objects which represent *PM_1* through *PM_8* are attached to the *SENSORS* hierarchy at the *PWR_SENSORS* node. To represent their role as transponder components which could potentially fail, each signal power level sensor is also represented FIDEX as a member of the component world; belonging to the class of *POWER_METERS*.

Code Segment 4.12b: *PWR_SENSOR* Objects

```
(@OBJECT=       PM_1
        (@CLASSES=
                POWER_METERS
                PWR_SENSORS     )       )

(@OBJECT=       PM_2
        (@CLASSES=
                POWER_METERS
                PWR_SENSORS     )       )

(@OBJECT=       PM_3
        (@CLASSES=
                POWER_METERS
                PWR_SENSORS     )       )

(@OBJECT=       PM_4
        (@CLASSES=
                POWER_METERS
                PWR_SENSORS     )       )

(@OBJECT=       PM_5
        (@CLASSES=
                POWER_METERS
                PWR_SENSORS     )       )

(@OBJECT=       PM_6
        (@CLASSES=
                POWER_METERS
                PWR_SENSORS     )       )

(@OBJECT=       PM_7
        (@CLASSES=
                POWER_METERS
                PWR_SENSORS    ·)       )

(@OBJECT=       PM_8
        (@CLASSES=
                POWER_METERS
                PWR_SENSORS     )       )
```

4.4 Representation of Fault States

The transponder fault states are represented as objects in a class structure called *FAULT_STATES*. This class is also divided into several subclasses. Each subclass frame represents the association of fault states to component types; such as amplifier faults, attenuator faults, etc. Object frames representing the specific failure modes of the transponder are then attached to the appropriate subclasses. This structure enables FIDEX to reason about both known and abstract faults.

The code segment which defines this structure is nearly identical to that of the *COMPONENTS* class. This is because the types of fault states are associated with the types of components.

4.4.1 Property Definitions

The properties associated with the *FAULT_STATES* class are listed in the next code segment. These describe which *COMPONENT* the fault is associated with, its *INFeRence CATEGORY* or priority, and the *POWER SYMPTOM GROUP* with which is associated. A Boolean property, *VERIFIED*, is used to flag fault states which have been verified by the diagnostic process. The final property listed is *Value*. This property is a reserved by NEXPERT™. The fault states represent the hypotheses of rules used during diagnosis. This property is assigned the results of rule evaluations.

Code Segment 4.13: Properties of the *FAULT_STATES* Class

```
(@PROPERTY=    COMPONENT               @TYPE=String;)
(@PROPERTY=    INF_CAT                 @TYPE=Float;)
(@PROPERTY=    POWER_SYMPTOM_GROUP     @TYPE=String;)
(@PROPERTY=    Value                   @TYPE=Special;)
(@PROPERTY=    VERIFIED                @TYPE=Boolean;)
```

In chapter 3, the *CERTAINTY_ANALYSIS* class was introduced. This is a superclass of the *FAULT_STATES* class. It is used to define the overhead required for inexact and abstract reasoning; as discussed in section 3.3. Code Segment 4.14 gives the definition of properties required for the *CERTAINTY_ANALYSIS* superclass.

Code Segment 4.14: Properties of the *CERTAINTY_ANALYSIS* Class

```
(@PROPERTY=     AB              @TYPE=Float;)
(@PROPERTY=     AD              @TYPE=Float;)
(@PROPERTY=     CF              @TYPE=Float;)
(@PROPERTY=     CONFIDENCE      @TYPE=String;)
(@PROPERTY=     MB              @TYPE=Float;)
(@PROPERTY=     MD              @TYPE=Float;)
```

Five floating point properties are used to represent each quantity in the MYCIN equations. The current measures of belief and disbelief are represented by the *MB* and *MD* properties. The accumulated belief and disbelief are represented by the *AB* and *AD* properties. Finally, the overall confidence is represented by the *CF* property. A string property called *CONFIDENCE* is used for a qualitative description of confidence.

4.4.2 Class Definitions

The definitions for classes in the *FAULT_STATES* hierarchy are given in the remaining code segments. First, the definition in Code Segment 4.15 creates the superclass for *CERTAINTY_ANALYSIS.* The *FAULT_STATES* class is attached as a subclass, and the properties discussed above are defined with this class.

Code Segment 4.15: Definition of *CERTAINTY_ANALYSIS* Class

```
(@CLASS=        CERTAINTY_ANALYSIS
        (@SUBCLASSES=   FAULT_STATES    )
        (@PROPERTIES=
                AB
                AD
                CF
                CONFIDENCE
                MB
                MD      )       )
```

Code Segment 4.16 defines the class for *FAULT_STATES* in the object space of the FIDEX system. The properties in Code Segment 4.13 are assigned and class for each type of fault state attached as subclasses. The definitions for the classes which represent these fault state types are given in Code Segment 4.17.

Code Segment 4.16: Definition of the *FAULT_STATES* Class

```
(@CLASS=          FAULT_STATES
        (@SUBCLASSES=
                  AMPLIFIER_FAULTS
                  ATTENUATOR_FAULTS
                  GaAs_FET_FAULTS
                  LO_FAULTS
                  MIXER_FAULTS
                  RECEIVER_FAULTS
                  SWITCH_FAULTS
                  TWTA_FAULTS     )        )
```

Each class defined in Code Segment 4.17 represents an association of a fault state with a type of transponder component. The class called *AMPLIFIER_FAULTS* is used to classify all fault states associated with *AMPLIFIER* components. The class called *ATTENUATOR_FAULTS* is used to classify all fault states associated with *ATTENUATOR* components. The class called *GaAs_FET_FAULTS* is used to classify all fault states associated with *GaAs_FET* components. The class called *LO_FAULTS* is used to classify all fault states associated with *LOCAL_OSCILLATOR* components. The class called *MIXER_FAULTS* is used to classify all fault states associated with *MIXER* components. The class called *RECEIVER_FAULTS* is used to classify all fault states associated with *RECEIVER* components. The class called *SWITCH_FAULTS* is used to classify all fault states associated with *SWITCH* components. And finally, the class called *TWTA_FAULTS* is used to classify all fault states associated with *TWTA* components.

Code Segment 4.17: Subclasses of the *FAULT_STATES* Hierarchy

```
(@CLASS=        AMPLIFIER_FAULTS          )
(@CLASS=        ATTENUATOR_FAULTS         )
(@CLASS=        GaAs_FET_FAULTS  )
(@CLASS=        LO_FAULTS         )
(@CLASS=        MIXER_FAULTS      )
(@CLASS=        RECEIVER_FAULTS  )
(@CLASS=        SWITCH_FAULTS     )
(@CLASS=        TWTA_FAULTS       )
```

The discussion of the objects which represent the fault states in this hierarchy is presented in later chapters. As each diagnostic module is presented, the fault states associated with that subsystem are discussed.

CHAPTER V

FIDEX KERNEL KNOWLEDGE BASE

This chapter continues discussion on the kernel of the frame-based knowledge of the FIDEX system. The previous chapter discussed the definition of classes, objects, and properties to represent the structure, operation, and fault states of the ACTS transponder system. In this chapter, the object dynamics of the FIDEX.tkb knowledge base are discussed.

5.1 Inference Strategies

The first topic of importance is the definition of the global inheritance and inference strategies used by the FIDEX system, see Code Segment 5.1. The first two definitions establish the global strategy for value inheritance within frame hierarchies. Upward value inheritance is disabled and downward value inheritance is enabled.

The next two definitions establish the inheritance strategies for property inheritance within an object/subobject hierarchy. These are only included for completeness. There is no property inheritance in the object/subobject hierarchies within the FIDEX system. All such inheritances, both upward and downward, are disabled.

The fifth and sixth definitions establish inheritance strategies within class hierarchies. As for the value inheritance, class properties are inherited downward only. The seventh definition in Code Segment 5.1 enables breadthwise inheritance through a lattice of hierarchies. This definition is very important. NEXPERT™'s default setting for this global is *FALSE*. It must be set to *TRUE* for the lattice structure of the FIDEX system to function properly. The eighth definition disables parent-to-child inheritance during run time. Setting this global to *FALSE* enables class level properties to be assigned values which are not inherited by its child objects. The importance of this is elaborated upon in the discussion of abstract fault states.

Code Segment 5.1: Global Inference Strategy Definitions

```
(@VERSION=      020)
(@GLOBALS=      @INHVALUP=FALSE;
                @INHVALDOWN=TRUE;
                @INHOBJUP=FALSE;
                @INHOBJDOWN=FALSE;
                @INHCLASSUP=FALSE;
                @INHCLASSDOWN=TRUE;
                @INHBREADTH=TRUE;
                @INHPARENT=FALSE;
                @PWTRUE=TRUE;
                @PWFALSE=TRUE;
                @PWNOTKNOWN=TRUE;
                @EXHBWRD=TRUE;
                @PTGATES=TRUE;
                @PFACTIONS=TRUE;
                @SOURCESON=TRUE;
                @CACTIONSON=TRUE;        )
```

The next six definitions establish the global strategy for propagation of inferencing. These definitions are changed periodically by certain methods. However, this global strategy is maintained throughout most of the knowledgebase. To enable foreword chaining, full propagation is required. Therefore, propagation while true, false, and notknown are enabled. Since foreword chaining in NEXPERT™ is accomplished through a mechanism called *gating*, propagation through gates must also be enabled. And finally, since many foreword chaining strategies are initiated from meta-slot actions, propagation from actions must also be enabled. The exhaustive backward strategy is enabled to allow foreword actions to evaluate contexts in which one hypothesis is supported by multiple rules.

The final two definitions in Code Segment 5.1 enable the order-of-sources (OS), *(@SOURCES=...)*, and if-change (IC) actions, *(@CACTIONS=...)*, within property slots. These are fundamental to the performance of the FIDEX system. Both must be set to *TRUE*.

5.2 Initialization of Object/Class Parameters

The values of many of the properties introduced in chapter 4 represent constant quantities. Such properties are those used to represent object names, input/output parameters, and nominal parameter values. This section discusses the initialization of these properties using both hard-coded and dynamic assignments.

Property values can be initialized through it meta-slots in two manners. The FIDEX kernel knowledge base uses both of these. The first way to initialize a property value is by using the *initial value*, *(@INITVAL=...)*, definition within a slot definition. When this method is used, the value of the slot is initialized to the defined value during the initialization of the knowledge base.

The second method is to include a *run time value* directive in the OS of the slot that is associated with an object or class property. This directive provides the sources for a constant value during the run time of the inference process.

5.2.1 Initialization of *COMPONENTS* Parameters

The properties associated with the *COMPONENTS* class were introduced in section 4.1 and defined in Code Segment 4.1. Several of these provide information on the structure of the transponder system or nominal values for other component parameters. Specifically, these properties are *COMPONENT_IN*, *COMPONENT_OUT*, *DESCRIPTION*, *NAME*, *NASA_ID,* as well as the *NOMINAL_* values for *FREQUENCY, FREQUENCY_IN, FREQUENCY_OUT, GAIN, POWER_IN,* and *POWER_OUT.*

Names of Component Objects

The slot definitions for initializing the name of *COMPONENTS* objects are given in Code Segments 5.2a through 5.2c. These definitions initialize the *NAME* property of each object that represents components of the transponder system. Both *initial value* and *run time value* techniques are used.

Code Segment 5.2a: Initialization of <|*COMPONENTS*|>.*NAME*

```
(@SLOT=          GAASFET.NAME
        (@INITVAL=      "GAASFET")
        (@SOURCES=      (RunTimeValue     ("GAASFET"))      )          )

(@SLOT=          HPAPC_AMP_1.NAME
        (@INITVAL=      "HPAPC_AMP_1")
        (@SOURCES=      (RunTimeValue     ("HPAPC_AMP_1"))         )          )

(@SLOT=          HPAPC_AMP_2.NAME
        (@INITVAL=      "HPAPC_AMP_2")
        (@SOURCES=      (RunTimeValue     ("HPAPC_AMP_2"))         )          )

(@SLOT=          HPAPC_ATTN_1.NAME
        (@INITVAL=      "HPAPC_ATTN_1")
        (@SOURCES=      (RunTimeValue     ("HPAPC_ATTN_1"))        )          )

(@SLOT=          HPAPC_ATTN_2.NAME
        (@INITVAL=      "HPAPC_ATTN_2")
        (@SOURCES=      (RunTimeValue     ("HPAPC_ATTN_2"))        )          )

(@SLOT=          HPAPC_ATTN_3.NAME
        (@INITVAL=      "HPAPC_ATTN_3")
        (@SOURCES=      (RunTimeValue     ("HPAPC_ATTN_3"))        )          )

(@SLOT=          HPAPC_ATTN_4.NAME
        (@INITVAL=      "HPAPC_ATTN_4")
        (@SOURCES=      (RunTimeValue     ("HPAPC_ATTN_4"))        )          )
```

Code Segment 5.2b: Initialization of <|*COMPONENTS*|>.*NAME*

```
(@SLOT=          IFPC_AMP_1.NAME
        (@INITVAL=      "IFPC_AMP_1")
        (@SOURCES=      (RunTimeValue     ("IFPC_AMP_1"))  )          )

(@SLOT=          IFPC_AMP_2.NAME
        (@INITVAL=      "IFPC_AMP_2")
        (@SOURCES=      (RunTimeValue     ("IFPC_AMP_2"))  )          )

(@SLOT=          IFPC_AMP_3.NAME
        (@INITVAL=      "IFPC_AMP_3")
        (@SOURCES=      (RunTimeValue     ("IFPC_AMP_3"))  )          )

(@SLOT=          IFPC_AMP_4.NAME
        (@INITVAL=      "IFPC_AMP_4")
        (@SOURCES=      (RunTimeValue     ("IFPC_AMP_4"))  )          )

(@SLOT=          IFPC_ATTN_1.NAME
        (@INITVAL=      "IFPC_ATTN_1")
        (@SOURCES=      (RunTimeValue     ("IFPC_ATTN_1")) )          )

(@SLOT=          IFPC_ATTN_2.NAME
        (@INITVAL=      "IFPC_ATTN_2")
        (@SOURCES=      (RunTimeValue     ("IFPC_ATTN_2")) )          )

(@SLOT=          IFPC_ATTN_3.NAME
        (@INITVAL=      "IFPC_ATTN_3")
        (@SOURCES=      (RunTimeValue     ("IFPC_ATTN_3")) )          )

(@SLOT=          IFPC_ATTN_4.NAME
        (@INITVAL=      "IFPC_ATTN_4")
        (@SOURCES=      (RunTimeValue     ("IFPC_ATTN_4")) )          )

(@SLOT=          MSWITCH.NAME
        (@INITVAL=      "MSWITCH")
        (@SOURCES=      (RunTimeValue     ("MSWITCH"))      )          )

(@SLOT=          MULT_1.NAME
        (@INITVAL=      "MULT_1")
        (@SOURCES=      (RunTimeValue     ("MULT_1"))       )          )

(@SLOT=          MULT_2.NAME
        (@INITVAL=      "MULT_2")
        (@SOURCES=      (RunTimeValue     ("MULT_2"))       )          )

(@SLOT=          RCVR_1.NAME
        (@INITVAL=      "RCVR_1")
        (@SOURCES=      (RunTimeValue     ("RCVR_1"))       )          )

(@SLOT=          RCVR_2.NAME
        (@INITVAL=      "RCVR_2")
        (@SOURCES=      (RunTimeValue     ("RCVR_2"))       )          )

(@SLOT=          RCVR_LO.NAME
        (@INITVAL=      "RCVR_LO")
        (@SOURCES=      (RunTimeValue     ("RCVR_LO"))      )          )
```

Code Segment 5.2c: Initialization of <|*COMPONENTS*|>.*NAME*

```
(@SLOT=          TWTA.NAME
        (@INITVAL=      "TWTA")
        (@SOURCES=      (RunTimeValue      ("TWTA"))        )        )

(@SLOT=          UPX_LO.NAME
        (@INITVAL=      "UPX_LO")
        (@SOURCES=      (RunTimeValue      ("UPX_LO"))      )        )
```

Descriptions of Component Objects

The slot definitions for initializing the descriptions of *COMPONENTS* objects are given in Code Segments 5.2a through 5.2c. These definitions initialize the *DESCRIPTION* property of each object that represents components of the transponder system.

Code Segment 5.3a: Initialization of <|*COMPONENTS*|>.*DESCRIPTION*

```
(@SLOT=          GAASFET.DESCRIPTION
        (@INITVAL=      "Gallium-Arsenide Field Effect Transistor Amplifier")
        (@SOURCES=      (RunTimeValue      ("Gallium-Arsenide Field Effect Transistor Amplifier"))       ))

(@SLOT=          HPAPC_AMP_1.DESCRIPTION
        (@INITVAL=      "Channel 1 HPAIPC Driver Amplifier")
        (@SOURCES=      (RunTimeValue      ("Channel 1 HPAIPC Driver Amplifier"))  )       )

(@SLOT=          HPAPC_AMP_2.DESCRIPTION
        (@INITVAL=      "Channel 2 HPAIPC Driver Amplifier")
        (@SOURCES=      (RunTimeValue      ("Channel 2 HPAIPC Driver Amplifier"))  )       )

(@SLOT=          HPAPC_ATTN_1.DESCRIPTION
        (@INITVAL=      "Channel 1 High Power Amplifier Input Attenuator")
        (@SOURCES=      (RunTimeValue      ("Channel 1 High Power Amplifier Input Attenuator"))          ))

(@SLOT=          HPAPC_ATTN_2.DESCRIPTION
        (@INITVAL=      "Channel 1 HPAIPC Driver Input Attenuator")
        (@SOURCES=      (RunTimeValue      ("Channel 1 HPAIPC Driver Input Attenuator"))     )         )

(@SLOT=          HPAPC_ATTN_3.DESCRIPTION
        (@INITVAL=      "Channel 2 HPAIPC Driver Input Attenuator")
        (@SOURCES=      (RunTimeValue      ("Channel 2 HPAIPC Driver Input Attenuator"))     )         )

(@SLOT=          HPAPC_ATTN_4.DESCRIPTION
        (@INITVAL=      "Channel 2 High Power Amplifier Input Attenuator")
        (@SOURCES=      (RunTimeValue      ("Channel 2 High Power Amplifier Input Attenuator"))          ))
```

Code Segment 5.3b: Initialization of <|*COMPONENTS*|>.*DESCRIPTION*

```
(@SLOT=          IFPC_AMP_1.DESCRIPTION
        (@INITVAL=       "Channel 1 Matrix Switch Input IFPC Amplifier")
        (@SOURCES=       (RunTimeValue      ("Channel 1 Matrix Switch Input IFPC Amplifier"))  )          )

(@SLOT=          IFPC_AMP_2.DESCRIPTION
        (@INITVAL=       "Channel 2 Matrix Switch Input IFPC Amplifier")
        (@SOURCES=       (RunTimeValue      ("Channel 2 Matrix Switch Input IFPC Amplifier"))  )          )

(@SLOT=          IFPC_AMP_3.DESCRIPTION
        (@INITVAL=       "Channel 1 Up-converter Input IFPC Amplifier")
        (@SOURCES=       (RunTimeValue      ("Channel 1 Up-converter Input IFPC Amplifier"))   )          )

(@SLOT=          IFPC_AMP_4.DESCRIPTION
        (@INITVAL=       "Channel 2 Up-converter Input IFPC Amplifier")
        (@SOURCES=       (RunTimeValue      ("Channel 2 Up-converter Input IFPC Amplifier"))   )          )

(@SLOT=          IFPC_ATTN_1.DESCRIPTION
        (@INITVAL=       "Channel 1 Matrix Switch Input IFPC Attenuator")
        (@SOURCES=       (RunTimeValue      ("Channel 1 Matrix Switch Input IFPC Attenuator"))  )         )

(@SLOT=          IFPC_ATTN_2.DESCRIPTION
        (@INITVAL=       "Channel 2 Matrix Switch Input IFPC Attenuator")
        (@SOURCES=       (RunTimeValue      ("Channel 2 Matrix Switch Input IFPC Attenuator"))  )         )

(@SLOT=          IFPC_ATTN_3.DESCRIPTION
        (@INITVAL=       "Channel 1 Up-converter Input IFPC Attenuator")
        (@SOURCES=       (RunTimeValue      ("Channel 1 Up-converter Input IFPC Attenuator"))  )          )

(@SLOT=          IFPC_ATTN_4.DESCRIPTION
        (@INITVAL=       "Channel 2 Up-converter Input IFPC Attenuator")
        (@SOURCES=       (RunTimeValue      ("Channel 2 Up-converter Input IFPC Attenuator"))  )          )

(@SLOT=          MSWITCH.DESCRIPTION
        (@INITVAL=       "Ford Matrix Switch")
        (@SOURCES=       (RunTimeValue      ("Ford Matrix Switch"))         )          )

(@SLOT=          MULT_1.DESCRIPTION
        (@INITVAL=       "Channel 1 Up-converter Mixer")
        (@SOURCES=       (RunTimeValue      ("Channel 1 Up-converter Mixer"))         )          )

(@SLOT=          MULT_2.DESCRIPTION
        (@INITVAL=       "Channel 2 Up-converter Mixer")
        (@SOURCES=       (RunTimeValue      ("Channel 2 Up-converter Mixer"))         )          )

(@SLOT=          RCVR_1.DESCRIPTION
        (@INITVAL=       "Channel 1 Receiver Unit")
        (@SOURCES=       (RunTimeValue      ("Channel 1 Receiver Unit"))  )          )

(@SLOT=          RCVR_2.DESCRIPTION
        (@INITVAL=       "Channel 2 Receiver Unit")
        (@SOURCES=       (RunTimeValue      ("Channel 2 Receiver Unit"))  )          )

(@SLOT=          RCVR_LO.DESCRIPTION
        (@INITVAL=       "Receiver Units Local Oscillator")
        (@SOURCES=       (RunTimeValue      ("Receiver Units Local Oscillator"))        )          )
```

Code Segment 5.3c: Initialization of <|*COMPONENTS*|>.*DESCRIPTION*

```
(@SLOT=          TWTA.DESCRIPTION
        (@PIERCED=      "Traveling Wave Tube Amplifier")
        (@SOURCES=      (Pierced   ("Traveling Wave Tube Amplifier"))        )         )

(@SLOT=          PERSUADE_LO.DESCRIPTION
        (@PIERCED=      "Up-converter Mixer Units Local Oscillator")
        (@SOURCES=      (Pierced   ("Up-converter Mixer Units Local Oscillator"))        )         )
```

Retrieval of Remaining Property Values from Database

Only the values of properties for *NAME* and *DESCRIPTION* were hard-coded into the frame structure of the *COMPONENTS* world. The values for the remainder of the initialized properties are retrieved from COMPONT.nxp database. This database is included in section A.2 of Appendix A. Code Segments 5.4a through 5.4c give the definitions for OS slot actions which retrieve these values. The source actions for each of these slots are identical. Therefore, only the first slot definition in Code Segment 5.4a are discussed.

Whenever the value any of these properties for an object in the *COMPONENTS* class is unknown, it is retrieved from a database. The first argument of the retrieve directive defines the name of the database to retrieve from. The type, or format, of the database is set to the NEXPERT™ DataBase type. Forward chaining is disabled so that changes to these property values do not affect the agenda. And, the retrieve unknown is set active; enabling unknown values to be retrieved from the database.

Code Segment 5.4a: Slot Actions to Retrieve & Initialize Properties of *COMPONENTS* Class

```
(@SLOT=          COMPONENTS.COMPONENT_IN
     (@SOURCES=       (Retrieve          ("COMPONT.nxp")                            \
                      (@TYPE=NXPDB;                                                 \
                       @FWRD=FALSE;                                                 \
                       @UNKNOWN=TRUE;                                               \
                       @PROPS=NAME, COMPONENT_IN, COMPONENT_OUT, NASA_ID,           \
                              NOMINAL_FREQUENCY, NOMINAL_FREQUENCY_IN,              \
                              NOMINAL_FREQUENCY_OUT, NOMINAL_GAIN,                  \
                              NOMINAL_POWER_IN, NOMINAL_POWER_OUT;                  \
                       @FIELDS="NAME", "COMPONENT_IN", "COMPONENT_OUT", "NASA_ID",  \
                              "NOM_FREQ", "NOM_FREQ_IN", "NOM_FREQ_OUT",            \
                              "NOM_GAIN", "NOM_POWER_IN", "NOM_POWER_OUT";          \
                       @ATOMS=SELF;))        )        )

(@SLOT=          COMPONENTS.COMPONENT_OUT
     (@SOURCES=       (Retrieve          ("COMPONT.nxp")                            \
                      (@TYPE=NXPDB;                                                 \
                       @FWRD=FALSE;                                                 \
                       @UNKNOWN=TRUE;                                               \
                       @PROPS=NAME, COMPONENT_IN, COMPONENT_OUT, NASA_ID,           \
                              NOMINAL_FREQUENCY, NOMINAL_FREQUENCY_IN,              \
                              NOMINAL_FREQUENCY_OUT, NOMINAL_GAIN,                  \
                              NOMINAL_POWER_IN, NOMINAL_POWER_OUT;                  \
                       @FIELDS="NAME", "COMPONENT_IN", "COMPONENT_OUT", "NASA_ID",  \
                              "NOM_FREQ", "NOM_FREQ_IN", "NOM_FREQ_OUT",            \
                              "NOM_GAIN", "NOM_POWER_IN", "NOM_POWER_OUT";          \
                       @ATOMS=SELF;))        )        )

(@SLOT=          COMPONENTS.NASA_ID
     (@SOURCES=       (Retrieve          ("COMPONT.nxp")                            \
                      (@TYPE=NXPDB;                                                 \
                       @FWRD=FALSE;                                                 \
                       @UNKNOWN=TRUE;                                               \
                       @PROPS=NAME, COMPONENT_IN, COMPONENT_OUT, NASA_ID,           \
                              NOMINAL_FREQUENCY, NOMINAL_FREQUENCY_IN,              \
                              NOMINAL_FREQUENCY_OUT, NOMINAL_GAIN,                  \
                              NOMINAL_POWER_IN, NOMINAL_POWER_OUT;                  \
                       @FIELDS="NAME", "COMPONENT_IN", "COMPONENT_OUT", "NASA_ID",  \
                              "NOM_FREQ", "NOM_FREQ_IN", "NOM_FREQ_OUT",            \
                              "NOM_GAIN", "NOM_POWER_IN", "NOM_POWER_OUT";          \
                       @ATOMS=SELF;))        )        )

(@SLOT=          COMPONENTS.NOMINAL_FREQUENCY
     (@SOURCES=       (Retrieve          ("COMPONT.nxp")                            \
                      (@TYPE=NXPDB;                                                 \
                       @FWRD=FALSE;                                                 \
                       @UNKNOWN=TRUE;                                               \
                       @PROPS=NAME, COMPONENT_IN, COMPONENT_OUT, NASA_ID,           \
                              NOMINAL_FREQUENCY, NOMINAL_FREQUENCY_IN,              \
                              NOMINAL_FREQUENCY_OUT, NOMINAL_GAIN,                  \
                              NOMINAL_POWER_IN, NOMINAL_POWER_OUT;                  \
                       @FIELDS="NAME", "COMPONENT_IN", "COMPONENT_OUT", "NASA_ID",  \
                              "NOM_FREQ", "NOM_FREQ_IN", "NOM_FREQ_OUT",            \
                              "NOM_GAIN", "NOM_POWER_IN", "NOM_POWER_OUT";          \
                       @ATOMS=SELF;))        )        )
```

Code Segment 5.4b: Slot Actions to Retrieve & Initialize Properties of *COMPONENTS* Class

```
(@SLOT=          COMPONENTS.NOMINAL_FREQUENCY_IN
     (@SOURCES=      (Retrieve          ("COMPONT.nxp")                          \
                     (@TYPE=NXPDB;                                               \
                      @FWRD=FALSE;                                               \
                      @UNKNOWN=TRUE;                                             \
                      @PROPS=NAME, COMPONENT_IN, COMPONENT_OUT, NASA_ID,         \
                             NOMINAL_FREQUENCY, NOMINAL_FREQUENCY_IN,            \
                             NOMINAL_FREQUENCY_OUT, NOMINAL_GAIN,                \
                             NOMINAL_POWER_IN, NOMINAL_POWER_OUT;                \
                      @FIELDS="NAME", "COMPONENT_IN", "COMPONENT_OUT", "NASA_ID", \
                             "NOM_FREQ", "NOM_FREQ_IN", "NOM_FREQ_OUT",          \
                             "NOM_GAIN", "NOM_POWER_IN", "NOM_POWER_OUT";        \
                      @ATOMS=SELF;))          )          )

(@SLOT=          COMPONENTS.NOMINAL_FREQUENCY_OUT
     (@SOURCES=      (Retrieve          ("COMPONT.nxp")                          \
                     (@TYPE=NXPDB;                                               \
                      @FWRD=FALSE;                                               \
                      @UNKNOWN=TRUE;                                             \
                      @PROPS=NAME, COMPONENT_IN, COMPONENT_OUT, NASA_ID,         \
                             NOMINAL_FREQUENCY, NOMINAL_FREQUENCY_IN,            \
                             NOMINAL_FREQUENCY_OUT, NOMINAL_GAIN,                \
                             NOMINAL_POWER_IN, NOMINAL_POWER_OUT;                \
                      @FIELDS="NAME", "COMPONENT_IN", "COMPONENT_OUT", "NASA_ID", \
                             "NOM_FREQ", "NOM_FREQ_IN", "NOM_FREQ_OUT",          \
                             "NOM_GAIN", "NOM_POWER_IN", "NOM_POWER_OUT";        \
                      @ATOMS=SELF;))          )          )

(@SLOT=          COMPONENTS.NOMINAL_GAIN
     (@SOURCES=      (Retrieve          ("COMPONT.nxp")                          \
                     (@TYPE=NXPDB;                                               \
                      @FWRD=FALSE;                                               \
                      @UNKNOWN=TRUE;                                             \
                      @PROPS=NAME, COMPONENT_IN, COMPONENT_OUT, NASA_ID,         \
                             NOMINAL_FREQUENCY, NOMINAL_FREQUENCY_IN,            \
                             NOMINAL_FREQUENCY_OUT, NOMINAL_GAIN,                \
                             NOMINAL_POWER_IN, NOMINAL_POWER_OUT;                \
                      @FIELDS="NAME", "COMPONENT_IN", "COMPONENT_OUT", "NASA_ID", \
                             "NOM_FREQ", "NOM_FREQ_IN", "NOM_FREQ_OUT",          \
                             "NOM_GAIN", "NOM_POWER_IN", "NOM_POWER_OUT";        \
                      @ATOMS=SELF;))          )          )

(@SLOT=          COMPONENTS.NOMINAL_POWER_IN
     (@SOURCES=      (Retrieve          ("COMPONT.nxp")                          \
                     (@TYPE=NXPDB;                                               \
                      @FWRD=FALSE;                                               \
                      @UNKNOWN=TRUE;                                             \
                      @PROPS=NAME, COMPONENT_IN, COMPONENT_OUT, NASA_ID,         \
                             NOMINAL_FREQUENCY, NOMINAL_FREQUENCY_IN,            \
                             NOMINAL_FREQUENCY_OUT, NOMINAL_GAIN,                \
                             NOMINAL_POWER_IN, NOMINAL_POWER_OUT;                \
                      @FIELDS="NAME", "COMPONENT_IN", "COMPONENT_OUT", "NASA_ID", \
                             "NOM_FREQ", "NOM_FREQ_IN", "NOM_FREQ_OUT",          \
                             "NOM_GAIN", "NOM_POWER_IN", "NOM_POWER_OUT";        \
                      @ATOMS=SELF;))          )          )
```

Code Segment 5.4c: Slot Actions to Retrieve & Initialize Properties of *COMPONENTS* Class

```
(@SLOT=          COMPONENTS.NOMINAL_POWER_OUT
        (@SOURCES=     (Retrieve              ("COMPONT.nxp")                                  \
                       (@TYPE=NXPDB;                                                            \
                        @FWRD=FALSE;                                                            \
                        @UNKNOWN=TRUE;                                                          \
                        @PROPS=NAME, COMPONENT_IN, COMPONENT_OUT, NASA_ID,                      \
                               NOMINAL_FREQUENCY, NOMINAL_FREQUENCY_IN,                         \
                               NOMINAL_FREQUENCY_OUT, NOMINAL_GAIN,                             \
                               NOMINAL_POWER_IN, NOMINAL_POWER_OUT;                             \
                        @FIELDS="NAME", "COMPONENT_IN", "COMPONENT_OUT", "NASA_ID",             \
                               "NOM_FREQ", "NOM_FREQ_IN", "NOM_FREQ_OUT",                       \
                               "NOM_GAIN", "NOM_POWER_IN", "NOM_POWER_OUT";                     \
                        @ATOMS=SELF;))          )         )
```

The next two parameters list the property names for which values are to be retrieved and a corresponding list of database field names. Notice that whenever any of these property values is pursued all of them are retrieved from the database. This was done to make the data accesses more efficient. Because the entire record is retrieved on the first access, only one database access is required for each *COMPONENTS* object. The final parameter of the retrieve directive lists the atoms for which the retrieve is effected. This is a class level definition of sources that are inherited by each *COMPONENTS* object. Defining the atom as *SELF* causes only the record that corresponds to the current object to be retrieved.

5.2.2 Initialization of *SUBSYSTEMS* Parameters

The properties associated with the *SUBSYSTEMS* class were introduced in section 4.2 and defined in Code Segment 4.6. Several of these provide information on the structure of the transponder subsystems or nominal values for other parameters. Specifically, these properties are *NAME, SENSOR_IN, SENSOR_OUT, SUBSYSTEM_IN,* and *SUBSYSTEM_OUT.*

Names of Subsystem Objects

The slot definitions for initializing the *NAME* property of *SUBSYSTEMS* objects are given in Code Segment 5.5. This segment also lists definitions for the dynamic objects which represent the channels through the matrix switch.

Code Segment 5.5: Initialization of <|*SUBSYSTEMS*|>.*NAME*

```
(@SLOT=          CH1AMP.NAME
        (@INITVAL=      "CH1AMP")
        (@SOURCES=      (RunTimeValue     ("CH1AMP"))      )        )

(@SLOT=          CH1RCVR.NAME
        (@INITVAL=      "CH1RCVR")
        (@SOURCES=      (RunTimeValue     ("CH1RCVR"))     )        )

(@SLOT=          CH1UPX.NAME
        (@INITVAL=      "CH1UPX")
        (@SOURCES=      (RunTimeValue     ("CH1UPX"))      )        )

(@SLOT=          CH2AMP.NAME
        (@INITVAL=      "CH2AMP")
        (@SOURCES=      (RunTimeValue     ("CH2AMP"))      )        )

(@SLOT=          CH2RCVR.NAME
        (@INITVAL=      "CH2RCVR")
        (@SOURCES=      (RunTimeValue     ("CH2RCVR"))     )        )

(@SLOT=          CH2UPX.NAME
        (@INITVAL=      "CH2UPX")
        (@SOURCES=      (RunTimeValue     ("CH2UPX"))      )        )

(@SLOT=          MSWITCH_CH11.NAME
        (@INITVAL=      "MSWITCH_11")
        (@SOURCES=      (RunTimeValue     ("MSWITCH_11")) )         )

(@SLOT=          MSWITCH_CH12.NAME
        (@INITVAL=      "MSWITCH_12")
        (@SOURCES=      (RunTimeValue     ("MSWITCH_12")) )         )

(@SLOT=          MSWITCH_CH21.NAME
        (@INITVAL=      "MSWITCH_CH21")
        (@SOURCES=      (RunTimeValue     ("MSWITCH_CH21"))        )        )

(@SLOT=          MSWITCH_CH22.NAME
        (@INITVAL=      "MSWITCH_CH22")
        (@SOURCES=      (RunTimeValue     ("MSWITCH_CH22"))        )        )
```

Linking of Subsystem Input/Output Properties

The slot definitions for initializing the input/output parameters of *SUBSYSTEMS* objects are given in Code Segment 5.6a through 5.6d. These definitions initialize the *SENSOR_IN, SENSOR_OUT, SUBSYSTEM_IN,* and *SUBSYSTEM_OUT* properties of each object that represents subsystems of the transponder system.

Code Segment 5.6a: Initialization of < |*SUBSYSTEMS*| >.*SENSOR_IN* / *_OUT*

```
(@SLOT=         CH1AMP.SENSOR_IN
        (@INITVAL=       "PM_5")
        (@SOURCES=       (RunTimeValue      ("PM_5"))          )          )

(@SLOT=         CH1AMP.SENSOR_OUT
        (@INITVAL=       "PM_7")
        (@SOURCES=       (RunTimeValue      ("PM_7"))          )          )

(@SLOT=         CH1RCVR.SENSOR_IN
        (@INITVAL=       "PM_0")
        (@SOURCES=       (RunTimeValue      ("PM_0"))          )          )

(@SLOT=         CH1RCVR.SENSOR_OUT
        (@INITVAL=       "PM_1")
        (@SOURCES=       (RunTimeValue      ("PM_1"))          )          )

(@SLOT=         CH1UPX.SENSOR_IN
        (@INITVAL=       "PM_3")
        (@SOURCES=       (RunTimeValue      ("PM_3"))          )          )

(@SLOT=         CH1UPX.SENSOR_OUT
        (@INITVAL=       "PM_5")
        (@SOURCES=       (RunTimeValue      ("PM_5"))          )          )

(@SLOT=         CH2AMP.SENSOR_IN
        (@INITVAL=       "PM_6")
        (@SOURCES=       (RunTimeValue      ("PM_6"))          )          )

(@SLOT=         CH2AMP.SENSOR_OUT
        (@INITVAL=       "PM_8")
        (@SOURCES=       (RunTimeValue      ("PM_8"))          )          )

(@SLOT=         CH2RCVR.SENSOR_IN
        (@INITVAL=       "PM_0")
        (@SOURCES=       (RunTimeValue      ("PM_0"))          )          )

(@SLOT=         CH2RCVR.SENSOR_OUT
        (@INITVAL=       "PM_2")
        (@SOURCES=       (RunTimeValue      ("PM_2"))          )          )

(@SLOT=         CH2UPX.SENSOR_IN
        (@INITVAL=       "PM_4")
        (@SOURCES=       (RunTimeValue      ("PM_4"))          )          )

(@SLOT=         CH2UPX.SENSOR_OUT
        (@INITVAL=       "PM_6")
        (@SOURCES=       (RunTimeValue      ("PM_6"))          )          )
```

Code Segment 5.6b: Initialization of < |*SUBSYSTEMS*| >.*SENSOR_IN / _OUT*

```
(@SLOT=         MSWITCH_CH11.SENSOR_IN
        (@INITVAL=      "PM_1")
        (@SOURCES=      (RunTimeValue    ("PM_1"))        )        )

(@SLOT=         MSWITCH_CH11.SENSOR_OUT
        (@INITVAL=      "PM_3")
        (@SOURCES=      (RunTimeValue    ("PM_3"))        )        )

(@SLOT=         MSWITCH_CH12.SENSOR_IN
        (@INITVAL=      "PM_1")
        (@SOURCES=      (RunTimeValue    ("PM_1"))        )        )

(@SLOT=         MSWITCH_CH12.SENSOR_OUT
        (@INITVAL=      "PM_4")
        (@SOURCES=      (RunTimeValue    ("PM_4"))        )        )

(@SLOT=         MSWITCH_CH21.SENSOR_IN
        (@INITVAL=      "PM_2")
        (@SOURCES=      (RunTimeValue    ("PM_2"))        )        )

(@SLOT=         MSWITCH_CH21.SENSOR_OUT
        (@INITVAL=      "PM_3")
        (@SOURCES=      (RunTimeValue    ("PM_3"))        )        )

(@SLOT=         MSWITCH_CH22.SENSOR_IN
        (@INITVAL=      "PM_2")
        (@SOURCES=      (RunTimeValue    ("PM_2"))        )        )

(@SLOT=         MSWITCH_CH22.SENSOR_OUT
        (@INITVAL=      "PM_4")
        (@SOURCES=      (RunTimeValue    ("PM_4"))        )        )
```

C-2

Code Segment 5.6c: Initialization of < |*SUBSYSTEMS*| >.*SUBSYSTEM_IN / _OUT*

```
(@SLOT=          CH1AMP.SUBSYSTEM_IN
        (@INITVAL=      "CH1UPX")
        (@SOURCES=      (RunTimeValue      ("CH1UPX"))       )       )

(@SLOT=          CH1AMP.SUBSYSTEM_OUT
        (@INITVAL=      "NONE")
        (@SOURCES=      (RunTimeValue      ("NONE"))         )       )

(@SLOT=          CH1RCVR.SUBSYSTEM_IN
        (@INITVAL=      "NONE")
        (@SOURCES=      (RunTimeValue      ("NONE"))         )       )

(@SLOT=          CH1RCVR.SUBSYSTEM_OUT
        (@INITVAL=      "MSWITCH")
        (@SOURCES=      (RunTimeValue      ("MSWITCH"))      )       )

(@SLOT=          CH1UPX.SUBSYSTEM_IN
        (@INITVAL=      "MSWITCH")
        (@SOURCES=      (RunTimeValue      ("MSWITCH"))      )       )

(@SLOT=          CH1UPX.SUBSYSTEM_OUT
        (@INITVAL=      "CH1AMP")
        (@SOURCES=      (RunTimeValue      ("CH1AMP"))       )       )

(@SLOT=          CH2AMP.SUBSYSTEM_IN
        (@INITVAL=      "CH2UPX")
        (@SOURCES=      (RunTimeValue      ("CH2UPX"))       )       )

(@SLOT=          CH2AMP.SUBSYSTEM_OUT
        (@INITVAL=      "NONE")
        (@SOURCES=      (RunTimeValue      ("NONE"))         )       )

(@SLOT=          CH2RCVR.SUBSYSTEM_IN
        (@INITVAL=      "NONE")
        (@SOURCES=      (RunTimeValue      ("NONE"))         )       )

(@SLOT=          CH2RCVR.SUBSYSTEM_OUT
        (@INITVAL=      "MSWITCH")
        (@SOURCES=      (RunTimeValue      ("MSWITCH"))      )       )

(@SLOT=          CH2UPX.SUBSYSTEM_IN
        (@INITVAL=      "MSWITCH")
        (@SOURCES=      (RunTimeValue      ("MSWITCH"))      )       )

(@SLOT=          CH2UPX.SUBSYSTEM_OUT
        (@INITVAL=      "CH2AMP")
        (@SOURCES=      (RunTimeValue      ("CH2AMP"))       )       )
```

Code Segment 5.6d: Initialization of < | *SUBSYSTEMS* | >.*SUBSYSTEM_IN* / *_OUT*

```
(@SLOT=          MSWITCH_CH11.SUBSYSTEM_IN
        (@INITVAL=       "CH1RCVR")
        (@SOURCES=       (RunTimeValue     ("CH1RCVR"))     )        )

(@SLOT=          MSWITCH_CH11.SUBSYSTEM_OUT
        (@INITVAL=       "CH1UPX")
        (@SOURCES=       (RunTimeValue     ("CH1UPX"))      )        )

(@SLOT=          MSWITCH_CH12.SUBSYSTEM_IN
        (@INITVAL=       "CH1RCVR")
        (@SOURCES=       (RunTimeValue     ("CH1RCVR"))     )        )

(@SLOT=          MSWITCH_CH12.SUBSYSTEM_OUT
        (@INITVAL=       "CH2UPX")
        (@SOURCES=       (RunTimeValue     ("CH2UPX"))      )        )

(@SLOT=          MSWITCH_CH21.SUBSYSTEM_IN
        (@INITVAL=       "CH2RCVR")
        (@SOURCES=       (RunTimeValue     ("CH2RCVR"))     )        )

(@SLOT=          MSWITCH_CH21.SUBSYSTEM_OUT
        (@INITVAL=       "CH1UPX")
        (@SOURCES=       (RunTimeValue     ("CH2UPX"))      )        )

(@SLOT=          MSWITCH_CH22.SUBSYSTEM_IN
        (@INITVAL=       "CH2RCVR")
        (@SOURCES=       (RunTimeValue     ("CH2RCVR"))     )        )

(@SLOT=          MSWITCH_CH22.SUBSYSTEM_OUT
        (@INITVAL=       "CH2UPX")
        (@SOURCES=       (RunTimeValue     ("CH2UPX"))      )        )
```

5.2.3 Initialization of *SENSORS* Parameters

The properties associated with the *SENSORS* class were introduced in section 4.3 and defined in Code Segment 4.10. Several of these provide information on nominal values and for other parameters. Specifically, these properties are *DESCRIPTION, NAME, NOMINAL, TOLERANCE, TYPE,* and *ZERO_LEVEL.* However, before the code segments that define these initializations can be discussed, another object must be introduced.

Recall from Figure 1.2 that there are no signal power level sensors at the input to the receiver units at the channel 1 and channel 2 inputs. Also in chapter 3, the concept of a subsystem for isolating faults was defined as a group of components between power sensors. Furthermore, the criteria for isolating a fault to find a subsystem who's input signal power level was *"GOOD"* and output signal power level was *"BAD."* This situation resulted in a conflict in defining the channel 1 and 2 receiver subsystems.

This conflict was resolved by creating a hypothetical signal power level sensor for the inputs to the channel 1 and channel 2 receiver subsystems. This sensor would always report a *"GOOD"* reading; as it must be assumed that the uplink signal to the transponder is within its parametric range. This hypothetical sensor was represented by creating an object called *PM_0* and initializing its *READING* and *LEVEL* properties to *"GOOD"* and *"OK"* respectively. Code Segment 5.7 gives these definitions.

Code Segment 5.7: Definition of Hypothetical Signal Power Level Sensor *PM_0*

```
(@OBJECT=        PM_0
        (@PROPERTIES=
                 LEVEL
                 NAME
                 READING          )          )

(@SLOT=          PM_0.LEVEL
        (@INITVAL=       "OK")
        (@SOURCES=       (RunTimeValue)     ("OK"))  )          )

(@SLOT=          PM_0.NAME
        (@INITVAL=       "PM_0")
        (@SOURCES=       (RunTimeValue)     ("PM_0"))           )          )

(@SLOT=          PM_0.READING
        (@INITVAL=       "GOOD")
        (@SOURCES=       (RunTimeValue)     ("GOOD"))           )          )
```

Names of Sensor Objects

The slot definitions for initializing the *NAME* property of *SENSORS* objects are given in Code Segments 5.8.

Code Segment 5.8: Initialization of <|*SENSORS*|>.*NAME*

```
(@SLOT=        BER_1.NAME
        (@INITVAL=      "BER_1")
        (@SOURCES=      (RunTimeValue      ("BER_1"))      )      )

(@SLOT=        BER_2.NAME
        (@INITVAL=      "BER_2")
        (@SOURCES=      (RunTimeValue      ("BER_2"))      )      )

(@SLOT=        BER_3.NAME
        (@INITVAL=      "BER_3")
        (@SOURCES=      (RunTimeValue      ("BER_3"))      )      )

(@SLOT=        BER_4.NAME
        (@INITVAL=      "BER_4")
        (@SOURCES=      (RunTimeValue      ("BER_4"))      )      )

(@SLOT=        BER_5.NAME
        (@INITVAL=      "BER_5")
        (@SOURCES=      (RunTimeValue      ("BER_5"))      )      )

(@SLOT=        BER_6.NAME
        (@INITVAL=      "BER_6")
        (@SOURCES=      (RunTimeValue      ("BER_6"))      )      )

(@SLOT=        PM_1.NAME
        (@INITVAL=      "PM_1")
        (@SOURCES=      (RunTimeValue      ("PM_1"))       )      )

(@SLOT=        PM_2.NAME
        (@INITVAL=      "PM_2")
        (@SOURCES=      (RunTimeValue      ("PM_2"))       )      )

(@SLOT=        PM_3.NAME
        (@INITVAL=      "PM_3")
        (@SOURCES=      (RunTimeValue      ("PM_3"))       )      )

(@SLOT=        PM_4.NAME
        (@INITVAL=      "PM_4")
        (@SOURCES=      (RunTimeValue      ("PM_4"))       )      )

(@SLOT=        PM_5.NAME
        (@INITVAL=      "PM_5")
        (@SOURCES=      (RunTimeValue      ("PM_5"))       )      )

(@SLOT=        PM_6.NAME
        (@INITVAL=      "PM_6")
        (@SOURCES=      (RunTimeValue      ("PM_6"))       )      )

(@SLOT=        PM_7.NAME
        (@INITVAL=      "PM_7")
        (@SOURCES=      (RunTimeValue      ("PM_7"))       )      )

(@SLOT=        PM_8.NAME
        (@INITVAL=      "PM_8")
        (@SOURCES=      (RunTimeValue      ("PM_8"))       )      )
```

Descriptions of Sensor Objects

The slot definitions for initializing the descriptions of *PWR_SENSORS* objects are given in Code Segment 5.9. These definitions initialize the *DESCRIPTION* property of each object that represents a signal power level sensor in the transponder system.

Code Segment 5.9: Initialization of <|*SENSORS*|>.*DESCRIPTION*

```
(@SLOT=         PM_1.DESCRIPTION
        (@INITVAL=      "Channel 1 Matrix Switch Input Signal Power Level Sensor")
        (@SOURCES=      (RunTimeValue      ("Channel 1 Matrix Switch Input Signal Power Level Sensor"))  ))

(@SLOT=         PM_2.DESCRIPTION
        (@INITVAL=      "Channel 2 Matrix Switch Input Signal Power Level Sensor")
        (@SOURCES=      (RunTimeValue      ("Channel 2 Matrix Switch Input Signal Power Level Sensor"))  ))

(@SLOT=         PM_3.DESCRIPTION
        (@INITVAL=      "Channel 1 Up-converter Input Signal Power Level Sensor")
        (@SOURCES=      (RunTimeValue      ("Channel 1 Up-converter Input Signal Power Level Sensor"))   ))

(@SLOT=         PM_4.DESCRIPTION
        (@INITVAL=      "Channel 2 Up-converter Input Signal Power Level Sensor")
        (@SOURCES=      (RunTimeValue      ("Channel 2 Up-converter Input Signal Power Level Sensor"))   ))

(@SLOT=         PM_5.DESCRIPTION
        (@INITVAL=      "Channel 1 HPA Input Signal Power Level Sensor")
        (@SOURCES=      (RunTimeValue      ("Channel 1 HPA Input Signal Power Level Sensor"))            ))

(@SLOT=         PM_6.DESCRIPTION
        (@INITVAL=      "Channel 2 HPA Input Signal Power Level Sensor")
        (@SOURCES=      (RunTimeValue      ("Channel 2 HPA Input Signal Power Level Sensor"))            ))

(@SLOT=         PM_7.DESCRIPTION
        (@INITVAL=      "Channel 1 HPA Output Signal Power Level Sensor")
        (@SOURCES=      (RunTimeValue      ("Channel 1 HPA Output Signal Power Level Sensor"))           ))

(@SLOT=         PM_8.DESCRIPTION
        (@INITVAL=      "Channel 2 HPA Output Signal Power Level Sensor")
        (@SOURCES=      (RunTimeValue      ("Channel 2 HPA Output Signal Power Level Sensor"))           ))
```

Types of Sensor Objects

The slot definitions for initializing the *TYPE* property of *SENSORS* objects are given in Code Segment 5.10. Each sensor type is initialized at the subclass level for signal *PWR_SENSORS* and data stream *BER_SENSORS.*

Code Segment 5.10: Initialization of <|*BER_ / PWR_SENSOR*|>.*TYPE*

```
(@SLOT=          BER_SENSORS.TYPE
        (@INITVAL=       "BER")
        (@SOURCES=       (RunTimeValue      ("BER")) )         )

(@SLOT=          PWR_SENSORS.TYPE
        (@INITVAL=       "PM")
        (@SOURCES=       (RunTimeValue      ("PM")) )          )
```

Retrieval of Remaining Property Values from Database

Only the values of the property *NAME* are hard-coded into the frame structure of the *SENSORS* world. The values for the remainder of the initialized properties are retrieved from SENSOR.nxp database. This database is included in section A.1 of Appendix A. Code Segment 5.11 gives the definitions for OS slot actions that retrieve these values.

Also associated with the slot for *SENSORS.NOMINAL* is an IC definition. This definition is not related to the initialization of parameters. It is only included here because it is attached to the listed slot. This action is required for the ToolBook™ interface and is discussed in that section of this chapter.

Code Segment 5.11: Slot Actions to Retrieve & Initialize Properties of the *SENSORS* Class

```
(@SLOT=         SENSORS.NAME
     (@SOURCES=      (Retrieve          ("SENSOR.nxp")                                          \
                     (@TYPE= NXPDB;                                                             \
                      @FWRD= FALSE;                                                             \
                      @UNKNOWN= TRUE;                                                           \
                      @PROPS= NAME, NOMINAL, TOLERANCE, ZERO_LEVEL;                             \
                      @FIELDS= "NAME", "NOMINAL", "TOLERANCE", "ZERO_LEVEL";                    \
                      @ATOMS= SELF;))          )        )

(@SLOT=         SENSORS.NOMINAL
     (@SOURCES=      (Retrieve          ("SENSOR.nxp")                                          \
                     (@TYPE= NXPDB;                                                             \
                      @FWRD= FALSE;                                                             \
                      @UNKNOWN= TRUE;                                                           \
                      @PROPS= NAME, NOMINAL, TOLERANCE, ZERO_LEVEL;                             \
                      @FIELDS= "NAME", "NOMINAL", "TOLERANCE", "ZERO_LEVEL";                    \
                      @ATOMS= SELF;))          )        )
     (@CACTIONS=     (Execute  ("ReturnNominalData")                                            \
                     (@ATOMID= SELF;                                                            \
                      @STRING="@V(@SELF.NOMINAL)";))          )        )

(@SLOT=         SENSORS.TOLERANCE
     (@SOURCES=      (Retrieve          ("SENSOR.nxp")                                          \
                     (@TYPE= NXPDB;                                                             \
                      @FWRD= FALSE;                                                             \
                      @UNKNOWN= TRUE;                                                           \
                      @PROPS= NAME, NOMINAL, TOLERANCE, ZERO_LEVEL;                             \
                      @FIELDS= "NAME", "NOMINAL", "TOLERANCE", "ZERO_LEVEL";                    \
                      @ATOMS= SELF;))          )        )

(@SLOT=         SENSORS.ZERO_LEVEL
     (@SOURCES=      (Retrieve          ("SENSOR.nxp")                                          \
                     (@TYPE= NXPDB;                                                             \
                      @FWRD= FALSE;                                                             \
                      @UNKNOWN= TRUE;                                                           \
                      @PROPS= NAME, NOMINAL, TOLERANCE, ZERO_LEVEL;                             \
                      @FIELDS= "NAME", "NOMINAL", "TOLERANCE", "ZERO_LEVEL";                    \
                      @ATOMS= SELF;))          )        )
```

5.2.4 Initialization of *FAULT_STATES* Parameters

There are many parameters of the *FAULT_STATES* hierarchy which have initialized values. However, these initializations are very specific to the knowledge of the individual diagnostic modules. They are not defined in the FIDEX kernel knowledge base. Therefore, they ware discussed when applicable in chapter 8.

5.3 Definition of Blackboard Objects

The strength of any frame-based expert system lies in the efficient encoding of rule knowledge. Its rules should be generic and operate on conditions that are germane rather than specific to certain instances. A common approach that is used to increase the efficiency of rule knowledge in frame-based systems is to use a structure called a *blackboard.*

By using a blackboard, rules can be written to operate on information posted in a global structure. The FIDEX system uses this approach. Its rules operate on properties associated with four objects. The definitions for these objects are given in Code Segment 5.12.

Code Segment 5.12: Definition of Blackboard Objects

```
(@OBJECT=       CURRENT_COMPONENT
        (@PROPERTIES=  NAME  )        )

(@OBJECT=       CURRENT_FAULT
        (@PROPERTIES=  NAME  )        )

(@OBJECT=       CURRENT_SENSOR
        (@PROPERTIES=  NAME  )        )

(@OBJECT=       CURRENT_SUBSYSTEM
        (@PROPERTIES=
                LEVEL_IN
                LEVEL_OUT
                NAME
                READING_IN
                READING_OUT
                SENSOR_IN
                SENSOR_OUT      )        )
```

5.4 Definition of Objects/Properties for Rule Hypotheses

This section, and the last section, are leading to a discussion of object/class dynamics and slot actions that begins in section 5.5. Several of these dynamics use rule knowledge. NEXPERT™ requires that all properties and objects used in a rule, whether as conditions or hypotheses, to be defined before the definition of the rule. To facilitate the discussion in the following section, all such definitions have been grouped together in Code Segment 5.13. The meanings of each object and property are discussed when appropriate in section 5.5.

Code Segment 5.13: Definition of Objects/Properties for Rule Hypotheses

```
(@PROPERTY=    BAD                          @TYPE=Boolean;)
(@PROPERTY=    Bad_Sensors                  @TYPE=Boolean;)
(@PROPERTY=    GOOD                         @TYPE=Boolean;)
(@PROPERTY=    HIGH                         @TYPE=Boolean;)
(@PROPERTY=    LOW                          @TYPE=Boolean;)
(@PROPERTY=    Nominal_Sensor_Data          @TYPE=Boolean;)
(@PROPERTY=    OK                           @TYPE=Boolean;)
(@PROPERTY=    ZERO                         @TYPE=Boolean;)

(@OBJECT=      Evaluate_Certainty_Factors
        (@PROPERTIES=   Value     @TYPE=Boolean;  )        )

(@OBJECT=      Model_Matrix_Switch_SubSystem
        (@PROPERTIES=   Value     @TYPE=Boolean;  )        )

(@OBJECT=      Return_BAD_Sensors
        (@PROPERTIES=   Value     @TYPE=Boolean;  )        )

(@OBJECT=      Return_Nominal_Sensor_Data
        (@PROPERTIES=   Value     @TYPE=Boolean;  )        )

(@OBJECT=      Sensor_Level_Description
        (@PROPERTIES=
               HIGH
               LOW
               OK
               ZERO     )         )

(@OBJECT=      Sensor_Reading_Description
        (@PROPERTIES=
               BAD
               GOOD     )         )

(@OBJECT=      TBK_Request
        (@PROPERTIES=
               Bad_Sensors
               Nominal_Sensor_Data          )         )
```

5.5 Object/Class Dynamics and Slot Actions

The values of many properties introduced in chapter 4 represent dynamic quantities. Such properties are those used to simulate the propagation of a signal through the transponder system, evaluate qualitative descriptions of parameter values, and drive the inference strategies of the expert system.

5.5.1 Dynamics and Slot Actions of the *COMPONENTS* Class

The properties associated with the *COMPONENTS* class were introduced in section 4.1 and defined in Code Segment 4.1. Several of these simulate the propagation of the communication signal through the transponder system. Specifically, these properties are *GAIN*, *POWER_IN*, *POWER_OUT*, and their *MODEL_* counterparts.

Propagation of Signal Power Levels Through Components

The slot definitions which simulate the propagation of signal power levels through the components of the transponder system are given in Code Segment 5.14. The three slots used in this simulation are <|*COMPONENTS*|>.*GAIN*, *POWER_IN*, and *POWER_OUT*. These are class level slot definitions which are inherited by all component objects in the *COMPONENTS* hierarchy.

The propagation of signal power levels is simulated using two different techniques for driving slot actions. The first and most useful approach is called *data-driven*. In this approach the changing of data at the input to a component is used to drive the evaluation of other component object property values.

Code Segment 5.14: Propagation of Signal Power Levels Through Components

```
(@SLOT=         COMPONENTS.GAIN
    (@SOURCES=     (Do    (SELF.POWER_OUT-SELF.POWER_IN)        (SELF.GAIN))     )
    (@CACTIONS=    (Do    (SELF.POWER_IN+SELF.GAIN)      (SELF.POWER_OUT))       ))

(@SLOT=         COMPONENTS.POWER_IN
    (@SOURCES=     (Do    (\SELF.COMPONENT_IN\.POWER_OUT)       (SELF.POWER_IN)))
    (@CACTIONS=    (Do    (SELF.POWER_IN+SELF.GAIN)      (SELF.POWER_OUT))       ))

(@SLOT=         COMPONENTS.POWER_OUT
    (@SOURCES=     (Do    (SELF.POWER_IN+SELF.GAIN)      (SELF.POWER_OUT))       )
    (@CACTIONS=    (Do    (SELF.POWER_OUT)       (\SELF.COMPONENT_OUT\.POWER_IN))))
```

Recall from section 4.1 that the signal power level at the input to a component is represented by the property *POWER_IN*. Whenever the value of this property changes, IC actions, *(@CACTIONS=...)*, are taken. The new value of *POWER_IN* is added with the current value of the component's *GAIN* property to calculate a new value for the signal power level at the output to a component. This new value is placed in the property that represents this quantity, *POWER_OUT*.

Changing this output power level again stimulates another IC action to be taken. Whenever the value of a component's *POWER_OUT* property changes, it posts this new value as the signal power level at the input to the next component in the transponder's signal path. (The negligible attenuation of component couplings are disregarded.) To do this, it must evaluate the string value of its *COMPONENT_OUT* property; which was initialized to the object name of the component object at its output in section 5.2.1.

Posting this new signal power level as the input power level of the next component stimulates the same sequence of IC actions in the properties of that component object. In this manner, the signal power levels through out the transponder system are simulated in the objects which represent the transponder components.

The propagation of signal power levels is also simulated using another technique for driving slot actions. This approach is called *source-driven*. It is useful when values of parameters are not known and must be calculated. In this approach the pursuit of a value for one parameter is used to drive the evaluation of other property values.

Whenever values for a component object's *GAIN* or *POWER_OUT* are required but unknown, OS actions, *(@SOURCES=...)*, provide the means for calculating them from other properties. Should the value of a component object's *POWER_IN* be required but unknown, OS actions provide the means for it to look to the output signal power level of its input component, *COMPONENT_IN*. Again, when these values are obtained, changed from unknown, their IC actions propagate the new values.

Propagation of Modeled Signal Power Levels Through Components

Also recall from section 4.1 that a primitive model-based reasoning overhead is included in the object dynamics of the FIDEX system. These dynamics are effected through the *MODEL_* prefixed properties listed in Code Segment 4.1. The propagation of these modeled signal power levels through components is implemented in the same manner as those discussed with Code Segment 5.14. The definitions for these slot actions are given in Code Segment 5.15.

Code Segment 5.15: Propagation of Modeled Signal Power Levels Through Components

```
(@SLOT=          COMPONENTS.MODEL_GAIN
          (@SOURCES=     (Do      (SELF.MODEL_POWER_OUT-SELF.MODEL_POWER_IN)  \
                         (SELF.MODEL_GAIN))      )
          (@CACTIONS=    (Do      (SELF.MODEL_POWER_IN+SELF.MODEL_GAIN)              \
                                  (SELF.MODEL_POWER_OUT))         )        )

(@SLOT=          COMPONENTS.MODEL_POWER_IN
          (@SOURCES=     (Do      (\SELF.COMPONENT_IN\.MODEL_POWER_OUT)              \
                                  (SELF.MODEL_POWER_IN)) )
          (@CACTIONS=    (Do      (SELF.MODEL_POWER_IN+SELF.MODEL_GAIN)              \
                                  (SELF.MODEL_POWER_OUT))         )        )

(@SLOT=          COMPONENTS.MODEL_POWER_OUT
          (@SOURCES=     (Do      (SELF.MODEL_POWER_IN+SELF.MODEL_GAIN)              \
                                  (SELF.MODEL_POWER_OUT))         )
          (@CACTIONS=    (Do      (SELF.MODEL_POWER_OUT)                             \
                                  (\SELF.COMPONENT_OUT\.MODEL_POWER_IN))      )      )
```

Propagation of Signal Power Levels Through MultiPort Components

As discussed in section 4.1, there are a number of multiple port components within the transponder. The parameters which represent signal quantities at these additional ports were represented in those component objects by the properties suffixed by *_2* in Code Segment 4.3. However, the existence of multiple ports does not complicate the simulated propagation the transponder signal. The techniques are simply expanded as shown in Code Segment 5.16.

This segment does not give a complete listing. The scope of such a listing would be of great length and very redundant. Code Segment 5.16 simply provides two examples of how the gain of the second channel through the matrix switch is simulated. For a complete listing, refer to Appendix A.

Code Segment 5.16: Propagation of Signal Power Levels Through Multiple Port Components

```
>>* INCOMPLETE *<<

(@SLOT=         SWITCHES.GAIN_2
        (@SOURCES=    (Do     (SELF.POWER_OUT_2-SELF.POWER_IN_2) (SELF.GAIN_2))        )
        (@CACTIONS=   (Do     (SELF.POWER_IN_2+SELF.GAIN_2)  (SELF.POWER_OUT_2))       ))

(@SLOT=         SWITCHES.MODEL_GAIN_2
        (@SOURCES=    (Do     (SELF.MODEL_POWER_OUT_2-SELF.MODEL_POWER_IN_2)\
                      (SELF.MODEL_GAIN_2))     )
        (@CACTIONS=   (Do     (SELF.MODEL_POWER_IN_2+SELF.MODEL_GAIN_2)     \
                      (SELF.MODEL_POWER_OUT_2))       )          )
```

5.5.2 Dynamics and Slot Actions of the *SUBSYSTEMS* Class

The properties associated with the *SUBSYSTEMS* class were introduced in section 4.2 and defined in Code Segment 4.6. Several of these are used to represent qualitative descriptions of signal power sensor *READING*s and *LEVEL*s throughout the transponder system. Another property is responsible for loading and initializing the *DIAGNOSTIC_MODULE* associated with the specific subsystems. And finally, recall that the various channels through matrix switch subsystem were modeled as a series of dynamic objects. This subsection discusses these dynamics of the *SUBSYSTEMS* class.

Dynamic Modeling of the Matrix Switch SubSystem

Table 1.2 listed four permutations of signal paths through the matrix switch. Correspondingly, there are four signal paths through the matrix switch subsystem. Each of these paths was represented by an object defined in Code Segments 4.9a and 4.9b. However at any given time, only two of these paths are propagating a signal through the transponder. Which two of those four is determined by the configuration of the matrix switch component.

To represent this configuration a string property called *CONFIG* was attached to the object that represents the matrix switch component, *MSWITCH.* This property is set to a character that represents a particular configuration of the switch. Code Segment 5.17 gives the definition for a slot which is used to obtain a value for this property.

Code Segment 5.17: Sources for Matrix Switch Configuration

```
(@SLOT=           MSWITCH.CONFIG
        (@SOURCES=      (Execute  ("RequestMatrixSwitchConfig"))          )        )
```

The OS action of this slot execute an external handler that is associated with the ToolBook™ Graphical User Interface (GUI). This handler ascertains a value for the switch configuration and place it in the *CONFIG* property of the matrix switch.

In the current phase of development of the transponder system, only two configurations of the matrix switch are possible. These are represented by either "*A*" or "*B.*" When the ToolBook™ handler sets the configuration property value, a hypothesis called *Model_Matrix_Switch_Subsystem* is placed on the agenda through a process called *gating.*

This hypothesis is supported by the two rules defined in Code Segment 5.18. Based on the value of the *MSWITCH.CONFIG* property, only one of these rules may fire. Each rule provides actions, *(@RHS=...),* that dynamically attaches two objects to the *SUBSYSTEMS* class. The single rule that fires attaches objects which represent active signal paths through the matrix switch as subsystems of the transponder. As such, these objects inherit all property values and slot actions associated with that class.

Code Segment 5.18: Rules That Model the Matrix Switch Signal Paths as *SUBSYSTEMS*

```
(@RULE=         RULE_012__MODEL_MATRIX_SWITCH
        (@LHS= (Is        (MSWITCH.CONFIG)        ("B"))    )
        (@HYPO=           Model_Matrix_Switch_SubSystem)
        (@RHS= (CreateObject       (MSWITCH_CH12) (|SUBSYSTEMS|))
                          (CreateObject       (MSWITCH_CH21) (|SUBSYSTEMS|)) ))

(@RULE=         RULE_011__MODEL_MATRIX_SWITCH
        (@LHS= (Is        (MSWITCH.CONFIG)        ("A"))    )
        (@HYPO=           Model_Matrix_Switch_SubSystem)
        (@RHS= (CreateObject       (MSWITCH_CH11) (|SUBSYSTEMS|))
                          (CreateObject       (MSWITCH_CH22) (|SUBSYSTEMS|)) ))
```

Qualitative Descriptions of Input/Output Sensor Quantities

The knowledge for isolating faults to the subsystems of the transponder is contained in another knowledge base, called ISOLATE.tkb. Furthermore, the knowledge bases required to diagnose faults within isolated subsystems are contained in still other files. Each of these modules uses information provided by several properties of the *SUBSYSTEMS* class. Therefore, the kernel knowledge base must provide a means for ascertaining all required information.

Isolation requires values for the *READING* reported by the sensors at the input and output of each subsystem. As discussed in section 4.2, these strings are to be placed in the *READING_IN* and *READING_OUT* properties of a subsystem. Diagnostics requires values for the *LEVEL* reported by the sensors at the input and output of each subsystem. As also discussed in section 4.2, these strings are to be placed in the *LEVEL_IN* and *LEVEL_OUT* properties of a subsystem. The slot actions listed in Code Segment 5.19 are defined at the *SUBSYSTEMS* class level and are used to obtain values for these properties from the *SENSORS* hierarchy.

Code Segment 5.19: Qualitative Descriptions for *SUBSYSTEMS* Input/Output Quantities

```
(@SLOT=         SUBSYSTEMS.LEVEL_IN
        (@SOURCES=    (Do      (\SELF.SENSOR_IN\.LEVEL) (SELF.LEVEL_IN)) )         )

(@SLOT=         SUBSYSTEMS.LEVEL_OUT
        (@SOURCES=    (Do      (\SELF.SENSOR_OUT\.LEVEL)        (SELF.LEVEL_OUT))       ))

(@SLOT=         SUBSYSTEMS.READING_IN
        (@SOURCES=    (Do      (\SELF.SENSOR_IN\.READING)       (SELF.READING_IN))      ))

(@SLOT=         SUBSYSTEMS.READING_OUT
        (@SOURCES=    (Do      (\SELF.SENSOR_OUT\.READING)      (SELF.READING_OUT))     ))
```

Recall from section 5.2.2 that names of the objects that represent the sensors at a subsystems input and output are initialized within the properties *SENSOR_IN* and *SENSOR_OUT*. Each slot action shown in Code Segment 5.19 provides the OS actions to evaluate these sensor names and acquire the qualitative descriptions as required.

Dynamics of an Isolated Subsystem Object

The job of the isolation module is to set the *ISOLATED* property of the objects that represents the subsystem that has been isolated. The knowledge required for this is discussed later in chapter 7. For now, the dynamics which occur when this property is set will be discussed.

Code Segment 5.20 defines a class level slot for the *ISOLATED* property of subsystems. During initialization, and at run time, the value of this property is set to *FALSE* for all objects in the *SUBSYSTEMS* class. The *run time value* directive in the OS actions assures that this is the case for the dynamically attached object that were discussed earlier.

Code Segment 5.20: Dynamics of < |*SUBSYSTEMS*| >.*ISOLATED*

```
(@SLOT=          SUBSYSTEMS.ISOLATED
      (@INITVAL=       FALSE)
      (@SOURCES=       (RunTimeValue      (FALSE)  )           )
      (@CACTIONS=      (Execute  ("ReturnIsolation")                          \
                       (@ATOMID=SELF;@STRING="@V(@SELF.NAME)";))    )         )
```

When the isolation module sets the *ISOLATED* property of a subsystem object to *TRUE*, a IC action is initiated. This action executes an external handler that is defined within the ToolBook™ GUI. Two parameters are passed to this handler. The first is the name of the object atom, *(@ATOMID=...)*, for the isolated subsystem. The key word *SELF* is used to pass the name of the subsystem object which triggered this IC action. Also passed is the value of the subsystem's *NAME* property. Both of these parameters are used within the ToolBook™ GUI.

When the GUI has finished informing the user of the results from the isolation phase, the handler for *ReturnIsolation* sets the *DIAGNOSTIC_MODULE* property of the subsystem object that called it. This is why the atom ID was passed as a parameter. Code Segment 5.21 lists the definitions for slots associated with this property for each object in the *SUBSYSTEMS* class. It is the IC actions associated with these slots which effect the chaining to the specialized diagnostic modules for the specific subsystems of the transponder.

Code Segment 5.21: Chaining to Diagnostic Modules

```
(@SLOT=         CH1AMP.DIAGNOSTIC_MODULE
     (@INITVAL=      FALSE)
     (@SOURCES=      (RunTimeValue     (FALSE)) )
     (@CACTIONS=     (LoadKB  ("CH1AMP.tkb")    (@LEVEL=ENABLE;))      )        )

(@SLOT=         CH1RCVR.DIAGNOSTIC_MODULE
     (@INITVAL=      FALSE)
     (@SOURCES=      (RunTimeValue     (FALSE)) )
     (@CACTIONS=     (LoadKB  ("CH1RCVR.tkb")   (@LEVEL=ENABLE;))      )        )

(@SLOT=         CH1UPX.DIAGNOSTIC_MODULE
     (@INITVAL=      FALSE)
     (@SOURCES=      (RunTimeValue     (FALSE)) )
     (@CACTIONS=     (LoadKB  ("CH1UPX.tkb")    (@LEVEL=ENABLE;))      )        )

(@SLOT=         CH2AMP.DIAGNOSTIC_MODULE
     (@INITVAL=      FALSE)
     (@SOURCES=      (RunTimeValue     (FALSE)) )
     (@CACTIONS=     (LoadKB  ("CH2AMP.tkb")    (@LEVEL=ENABLE;))      )        )

(@SLOT=         CH2RCVR.DIAGNOSTIC_MODULE
     (@INITVAL=      FALSE)
     (@SOURCES=      (RunTimeValue     (FALSE)) )
     (@CACTIONS=     (LoadKB  ("CH2RCVR.tkb")   (@LEVEL=ENABLE;))      )        )

(@SLOT=         CH2UPX.DIAGNOSTIC_MODULE
     (@INITVAL=      FALSE)
     (@SOURCES=      (RunTimeValue     (FALSE)) )
     (@CACTIONS=     (LoadKB  ("CH2UPX.tkb")    (@LEVEL=ENABLE;))      )        )

(@SLOT=         MSWITCH_CH11.DIAGNOSTIC_MODULE
     (@INITVAL=      FALSE)
     (@SOURCES=      (RunTimeValue     (FALSE)) )
     (@CACTIONS=     (LoadKB  ("MSWITCH.tkb")  (@LEVEL=ENABLE;))      )        )

(@SLOT=         MSWITCH_CH12.DIAGNOSTIC_MODULE
     (@INITVAL=      FALSE)
     (@SOURCES=      (RunTimeValue     (FALSE)) )
     (@CACTIONS=     (LoadKB  ("MSWITCH.tkb")  (@LEVEL=ENABLE;))      )        )

(@SLOT=         MSWITCH_CH21.DIAGNOSTIC_MODULE
     (@INITVAL=      FALSE)
     (@SOURCES=      (RunTimeValue     (FALSE)) )
     (@CACTIONS=     (LoadKB  ("MSWITCH.tkb")  (@LEVEL=ENABLE;))      )        )

(@SLOT=         MSWITCH_CH22.DIAGNOSTIC_MODULE
     (@INITVAL=      FALSE)
     (@SOURCES=      (RunTimeValue     (FALSE)) )
     (@CACTIONS=     (LoadKB  ("MSWITCH.tkb")  (@LEVEL=ENABLE;))      )        )
```

As for the *ISOLATED* property, the value of each object's *DIAGNOSTIC_MODULE* property is set to *FALSE* both during initialization and at run time. When the GUI handler sets this value to *TRUE* an IC action is stimulated. This action loads the specialized diagnostic knowledge base associated with the isolated subsystem object. As there is one diagnostic knowledge base for each subsystem, a slot must be defined at the object level for each subsystem object.

5.5.3 Dynamics and Slot Actions of the *SENSORS* Class

The properties associated with the *SENSORS* class were introduced in section 4.3 and defined in Code Segment 4.10. Several of these are used to ascertain qualitative descriptions of the data reported by the sensors of the transponder system. Specifically, these properties are *DATA, ERROR, LEVEL,* and *READING.*

Slot Actions for Qualitative Descriptions of Sensor Quantities

The slot definitions which ascertain qualitative descriptions of the data reported by sensor components are given in Code Segment 5.22. The values for sensor *DATA* are provided through the GUI. As these values change, IC actions defined at the class level drive the evaluation of qualitative discriptions for the sensor's *READING* and *LEVEL.* The first three actions associated with *DATA* reset the values of a sensors *ERROR, READING,* and *LEVEL* to unknown values. This is done to force their re-evaluation when the next three actions are taken. Assigning the value of a slot to itself, as done in the last three IC actions associated with *DATA,* is an efficient method for forcing the evaluation of a slot. Neuron Data, Inc., the publishers of NEXPERT™, recommends this technique for situations such as this one.

Code Segment 5.22: Slot Actions for Qualitative Descriptions of Sensor Quantities

```
(@SLOT=         SENSORS.DATA
        (@CACTIONS=     (Reset      (SELF.ERROR))
                        (Reset      (SELF.READING))
                        (Reset      (SELF.LEVEL))
                        (Do         (SELF.ERROR)        (SELF.ERROR))
                        (Do         (SELF.READING)      (SELF.READING))
                        (Do         (SELF.LEVEL)        (SELF.LEVEL))       )       )

(@SLOT=         SENSORS.ERROR .
        (@SOURCES=      (Do         (SELF.DATA-SELF.NOMINAL)        (SELF.ERROR))       )       )

(@SLOT=         SENSORS.READING
        (@SOURCES=      (Do         (SELF.NAME)         (CURRENT_SENSOR.NAME))
                        (Reset      (Sensor_Reading_Description.BAD))
                        (Do         (Sensor_Reading_Description.BAD)                            \
                                    (Sensor_Reading_Description.BAD))
                        (Reset      (Sensor_Reading_Description.GOOD))
                        (Do         (Sensor_Reading_Description.GOOD)                           \
                                    (Sensor_Reading_Description.GOOD))      )
        (@CACTIONS=     (Execute    ("ReturnSensorReading")                                     \
                        (@ATOMID=SELF;@STRING="@V(@SELF.READING)";)) )          )

(@SLOT=         SENSORS.LEVEL
        (@SOURCES=      (Do         (SELF.NAME)         (CURRENT_SENSOR.NAME))
                        (Reset      (Sensor_Level_Description.ZERO))
                        (Do         (Sensor_Level_Description.ZERO)                             \
                                    (Sensor_Level_Description.ZERO))
                        (Reset      (Sensor_Level_Description.LOW))
                        (Do         (Sensor_Level_Description.LOW)                              \
                                    (Sensor_Level_Description.LOW))
                        (Reset      (Sensor_Level_Description.HIGH))
                        (Do         (Sensor_Level_Description.HIGH)                             \
                                    (Sensor_Level_Description.HIGH))
                        (Reset      (Sensor_Level_Description.OK))
                        (Do         (Sensor_Level_Description.OK)                               \
                                    (Sensor_Level_Description.OK))          )
        (@CACTIONS=     (Execute    ("ReturnSensorLevel")                                       \
                        (@ATOMID=SELF;@STRING="@V(@SELF.LEVEL)";))    )         )
```

The value of the current sensors's error, *SELF.ERROR*, is unknown at this time. When its value is evaluated, the OS actions associated with the class level slot definition calculate the difference between the current and nominal sensor data values as the signed error in the sensor reading. This value is used in determining values for the reading and level. It is discussed shortly.

The sources for *READING* and *LEVEL* perform very similar functions. When values for each of these are pursued, the OS actions do the following. First, the value for *NAME* of the blackboard object *CURRENT_SENSOR* is set to the name of the current sensor object. Then a sequence of slots are reset and then forced to be evaluated. These slots represent the hypotheses of rules which ascribe the qualitative descriptions to the sensor quantities in question. This list is pursued sequentially until a value is set. Once a value is set, the OS actions are terminated and the remainder of the directive ignored. The rules which ascribe qualitative descriptions to sensor readings and levels are defined in Code Segments 5.23 and 5.24.

Also defined with the slots for *SENSORS.READING* and *.LEVEL* are IC actions which fire when the sources establish their values. These actions communicate the new qualitative descriptions to an external handler that is defined in the ToolBook™ GUI.

Code Segment 5.23: Rules to Ascribe Qualitative Descriptions to Sensor Levels

```
(@RULE=         RULE_003__QUALIFICATION_OF_HIGH_SENSOR_LEVELS
        (@LHS= (>       (\CURRENT_SENSOR.NAME\.ERROR) (0))       )
        (@HYPO=         Sensor_Level_Description.HIGH)
        (@RHS= (Let     (\CURRENT_SENSOR.NAME\.LEVEL) ("HIGH"))          )        )

(@RULE=         RULE_004__QUALIFICATION_OF_LOW_SENSOR_LEVELS
        (@LHS= (<       (\CURRENT_SENSOR.NAME\.ERROR) (0))       )
        (@HYPO=         Sensor_Level_Description.LOW)
        (@RHS= (Let     (\CURRENT_SENSOR.NAME\.LEVEL) ("LOW"))           )        )

(@RULE=         RULE_005__QUALIFICATION_OF_OK_SENSOR_LEVELS
        (@LHS= (<=      (ABS(\CURRENT_SENSOR.NAME\.ERROR)-                                   \
                        \CURRENT_SENSOR.NAME\.TOLERANCE)      (0))       )
        (@HYPO=         Sensor_Level_Description.OK)
        (@RHS= (Let     (\CURRENT_SENSOR.NAME\.LEVEL) ("OK"))  )         )

(@RULE=         RULE_006__QUALIFICATION_OF_ZERO_SENSOR_LEVELS
        (@LHS= (<=      (\CURRENT_SENSOR.NAME\.DATA-                              \
                        \CURRENT_SENSOR.NAME\.ZERO_LEVEL)     (0))       )
        (@HYPO=         Sensor_Level_Description.ZERO)
        (@RHS= (Let     (\CURRENT_SENSOR.NAME\.LEVEL) ("ZERO"))          )        )
```

Code Segment 5.24: Rules to Ascribe Qualitative Descriptions to Sensor Readings

```
(@RULE=         RULE_001__QUALIFICATION_OF_BAD_SENSOR_READINGS
        (@LHS= (>       (ABS(\CURRENT_SENSOR.NAME\.ERROR)-                                   \
                        \CURRENT_SENSOR.NAME\.TOLERANCE)      (0))       )
        (@HYPO=         Sensor_Reading_Description.BAD)
        (@RHS= (Let     (\CURRENT_SENSOR.NAME\.READING)          ("BAD"))
                        (CreateObject       (\CURRENT_SENSOR.NAME\)          (|BAD_SENSORS|))
                        (Let      (\CURRENT_SENSOR.NAME\.EVALUATED)      (TRUE)) ))

(@RULE=         RULE_002__QUALIFICATION_OF_GOOD_SENSOR_READINGS
        (@LHS= (<=      (ABS(\CURRENT_SENSOR.NAME\.ERROR)-                                   \
                        \CURRENT_SENSOR.NAME\.TOLERANCE)      (0))       )
        (@HYPO=         Sensor_Reading_Description.GOOD)
        (@RHS= (Let     (\CURRENT_SENSOR.NAME\.READING)          ("GOOD"))
                        (Let      (\CURRENT_SENSOR.NAME\.EVALUATED)      (TRUE)) ))
```

5.5.4 Implementation of MYCIN Technique and Certainty Analysis

The MYCIN technique for the incremental accumulation of evidence and its use in certainty analysis was first introduced in sections 3.3.2 and 3.3.3. Then, the representation of properties for certainty analysis were given in Code Segment 4.14.

This section discusses the implementation of the MYCIN technique at the *CERTAINTY_ANALYSIS* class level.

The slot definitions for the properties involved in certainty analysis are given in Code Segment 5.25. Supporting rules for ascribing qualitative descriptions to *CONFIDENCE* in fault state hypothesis are listed in Code Segment 5.26.

Code Segment 5.25: certainty analysis

```
(@SLOT= CERTAINTY_ANALYSIS.AB
        (@INITVAL=         0.0)
        (@SOURCES=         (RunTimeValue      (0.0))      )
        (@CACTIONS=        (Do        ((SELF.AB-SELF.AD)/MIN(SELF.AB,SELF.AD))          \
                                      (SELF.CF))             )           )

(@SLOT= CERTAINTY_ANALYSIS.AD
        (@INITVAL=         0.0)
        (@SOURCES=         (RunTimeValue      (0.0))      )
        (@CACTIONS=        (Do        ((SELF.AB-SELF.AD)/MIN(SELF.AB,SELF.AD))          \
                                      (SELF.CF))             )           )

(@SLOT= CERTAINTY_ANALYSIS.CF
        (@INITVAL=         0.0)
        (@SOURCES=         (RunTimeValue      (0.0))      )
        (@CACTIONS=        (Do        (SELF.NAME)         (CURRENT_FAULT.NAME))
                           (Reset     (Evaluate_Certainty_Factors))
                           (Do        (Evaluate_Certainty_Factors)          \
                                      (Evaluate_Certainty_Factors))         )          )

(@SLOT= CERTAINTY_ANALYSIS.MB
        (@INITVAL=         0.0)
        (@SOURCES=         (RunTimeValue      (0.0))      )
        (@CACTIONS=        (Do        (SELF.AB+SELF.MB*(1-SELF.AB))           (SELF.AB))
                           (Reset     (SELF.MB))             )           )

(@SLOT= CERTAINTY_ANALYSIS.MD
        (@INITVAL=         0.0)
        (@SOURCES=         (RunTimeValue      (0.0))      )
        (@CACTIONS=        (Do        (SELF.AD+SELF.MD*(1-SELF.AD))           (SELF.AD))
                           (Reset     (SELF.MD))             )           )
```

Code Segment 5.26: Rules to Ascribe Qualitative Descriptions to Fault State Confidence Factors

```
(@RULE=         RULE_029__QUALIFICATION_OF_CONFIDENCE__REJECTED
        (@LHS=  (<=     (\CURRENT_FAULT.NAME\.CF)        (-0.9))      )
        (@HYPO=         Evaluate_Certainty_Factors)
        (@RHS=  (Let    (\CURRENT_FAULT.NAME\.CONFIDENCE)        ("REJECTED"))
                        (Let      (\CURRENT_FAULT.NAME\.VERIFIED)  (FALSE))  )        )

(@RULE=         RULE_028__QUALIFICATION_OF_CONFIDENCE__VERY_IMPROBABLE
        (@LHS=  (<=     (\CURRENT_FAULT.NAME\.CF)        (-0.75))
                        (>        (\CURRENT_FAULT.NAME\.CF)        (-0.9))     )
        (@HYPO=         Evaluate_Certainty_Factors)
        (@RHS=  (Let    (\CURRENT_FAULT.NAME\.CONFIDENCE)                  \
                                  ("VERY_IMPROBABLE"))       )        )

(@RULE=         RULE_027__QUALIFICATION_OF_CONFIDENCE__IMPROBABLE
        (@LHS=  (<=     (\CURRENT_FAULT.NAME\.CF)        (-0.5))
                        (>        (\CURRENT_FAULT.NAME\.CF)        (-0.75))    )
        (@HYPO=         Evaluate_Certainty_Factors)
        (@RHS=  (Let    (\CURRENT_FAULT.NAME\.CONFIDENCE) ("IMPROBABLE"))  ))

(@RULE=         RULE_026__QUALIFICATION_OF_CONFIDENCE__UNLIKELY
        (@LHS=  (<=     (\CURRENT_FAULT.NAME\.CF)        (-0.25))
                        (>        (\CURRENT_FAULT.NAME\.CF)        (-0.5))     )
        (@HYPO=         Evaluate_Certainty_Factors)
        (@RHS=  (Let    (\CURRENT_FAULT.NAME\.CONFIDENCE)        ("UNLIKELY"))      ))

(@RULE=         RULE_025__QUALIFICATION_OF_CONFIDENCE__UNKNOWN
        (@LHS=  (>      (\CURRENT_FAULT.NAME\.CF)        (-0.25))
                        (<        (\CURRENT_FAULT.NAME\.CF)        (0.25))     )
        (@HYPO=         Evaluate_Certainty_Factors)
        (@RHS=  (Let    (\CURRENT_FAULT.NAME\.CONFIDENCE)        ("UNKNOWN"))       ))

(@RULE=         RULE_024__QUALIFICATION_OF_CONFIDENCE__POSSIBLE
        (@LHS=  (>=     (\CURRENT_FAULT.NAME\.CF)        (0.25))
                        (<        (\CURRENT_FAULT.NAME\.CF)        (0.5))      )
        (@HYPO=         Evaluate_Certainty_Factors)
        (@RHS=  (Let    (\CURRENT_FAULT.NAME\.CONFIDENCE)        ("POSSIBLE"))      ))

(@RULE=         RULE_023__QUALIFICATION_OF_CONFIDENCE__LIKELY
        (@LHS=  (>=     (\CURRENT_FAULT.NAME\.CF)        (0.5))
                        (<        (\CURRENT_FAULT.NAME\.CF)        (0.75))     )
        (@HYPO=         Evaluate_Certainty_Factors)
        (@RHS=  (Let    (\CURRENT_FAULT.NAME\.CONFIDENCE)        ("LIKELY"))        ))

(@RULE=         RULE_022__QUALIFICATION_OF_CONFIDENCE__PROBABLE
        (@LHS=  (>=     (\CURRENT_FAULT.NAME\.CF)        (0.75))
                        (<        (\CURRENT_FAULT.NAME\.CF)        (0.9))      )
        (@HYPO=         Evaluate_Certainty_Factors)
        (@RHS=  (Let    (\CURRENT_FAULT.NAME\.CONFIDENCE)        ("PROBABLE"))      ))

(@RULE=         RULE_021__QUALIFICATION_OF_CONFIDENCE__ESTABLISHED
        (@LHS=  (>=     (\CURRENT_FAULT.NAME\.CF)        (0.9))       )
        (@HYPO=         Evaluate_Certainty_Factors)
        (@RHS=  (Let    (\CURRENT_FAULT.NAME\.CONFIDENCE)        ("ESTABLISHED"))
                        (Let      (\CURRENT_FAULT.NAME\.VERIFIED)  (TRUE))  )        )
```

5.5.5 Dynamics of Interaction with ToolBook™ Interface

There are three properties of the *SENSORS* class that are used to communicate information to the ToolBook™ GUI. These properties are *RTN_LEVEL*, *RTN_NOMINAL*, and *RTN_READING*. They were introduced in section 4.3 and defined in Code Segment 4.10, but their descussion was delayed to now.

Their operation is simple. Whenever the GUI needs the current value of a sensors *LEVEL*, *NOMINAL*, or *READING* property, it can toggle the values of these slots between *TRUE* and *FALSE*. The IC actions given in Code Segment 5.27 are then stimulated to execute externally defined handlers and return the desired quantity.

Code Segment 5.27: Definition of GUI Interactive Slots

```
(@SLOT=         SENSORS.RTN_LEVEL
      (@CACTIONS=     (Execute  ("ReturnSensorLevel")                          \
                      (@ATOMID=SELF;@STRING="@V(@SELF.LEVEL)";))    )          )

(@SLOT=         SENSORS.RTN_NOMINAL
      (@CACTIONS=     (Execute  ("ReturnNominalData")                          \
                      (@ATOMID=SELF;@STRING="@V(@SELF.NOMINAL)";))             ))

(@SLOT=         SENSORS.RTN_READING
      (@CACTIONS=     (Execute  ("ReturnSensorReading")                        \
                      (@ATOMID=SELF;@STRING="@V(@SELF.READING)";)) )           )
```

A similar slot was defined earlier in Code Segment 5.11 for the IC actions of *SENSORS.NOMINAL*. It is a redundancy of the *SENSORS.RTN_NOMINAL* slot in that it communicates the value of the current sensor objects nominal data to the GUI. However, this redundancy added efficiency in speed and was included with the slot definition for the *NOMINAL* property slot.

The final code segment discussed in this chapter defines two rules which are required for the GUI. *RULE_901* is used to retrieve and return nominal sensor data to the

GUI. *RULE_902* is used to a list of all sensors who's readings were described as *"BAD."* Each of these use the IC actions defined in Code Segment 5.27 by toggling sensor property values.

Code Segment 5.28: Rules Required by ToolBook™ Interface

```
(@RULE=         RULE_902__RETURN_LIST_OF_BAD_SENSORS_TO_ToolBook
      (@LHS=  (Yes      (TBK_Request.Bad_Sensors))      )
      (@HYPO=           Return_BAD_Sensors)
      (@RHS=  (Let      ({|BAD_SENSORS|}.RTN_READING)      (TRUE))
                        (Strategy   (@CACTIONSON=FALSE;))
                        (Reset      ({|BAD_SENSORS|}.RTN_READING))
                        (Strategy   (@CACTIONSON=TRUE;))
                        (Execute    ("BadSensorReadingsReturned"))   )          )

(@RULE=         RULE_901__RETRIEVE_SENSOR_PARAMETERS_FROM_SENSOR_DATABASE_                \
                AND_RETURN_NOMINAL_DATA_TO_ToolBook
      (@LHS=  (Yes      (TBK_Request.Nominal_Sensor_Data))
                        (Retrieve    ("SENSOR.nxp")                                        \
                        (@TYPE=NXPDB;@FWRD=FALSE;@UNKNOWN=TRUE;@PROPS=NAME,                \
                        NOMINAL,TOLERANCE,ZERO_LEVEL;@FIELDS="NAME",                        \
                        "NOMINAL","TOLERANCE","ZERO_LEVEL";@ATOMS=<|SENSORS|>;))            )
      (@HYPO=           Return_Nominal_Sensor_Data)
      (@RHS=  (Do       (<|SENSORS|>.NOMINAL)     (<|SENSORS|>.NOMINAL))    ))
```

CHAPTER VI
FAULT DETECTION MODULE KNOWLEDGE BASE

This chapter discusses the rule module for fault detection knowledge. Recall from chapter 1 that each task in the diagnostic process was separated into an individual knowledge base. The knowledge for the detection of faults in the ACTS transponder system is contained in the DETECT.tkb knowledge base. A complete listing of that module can be found in Appendix B.

In section 1.3, the purpose of the fault detection task was defined as to detect any misbehavior in the performance of the ACTS transponder system. It was explained how this task involved the analysis of current sensor information in the form of qualitative descriptions for sensor readings. The knowledge required to detect a fault was then summarized as establishing a *"BAD"* reading on any sensor in the transponder.

The knowledge required to detect a fault, or determine that the transponder system is functioning properly is defined in Code Segment 6.1. First, the objects which represent the hypotheses of rules are defined. Then the two rules of the fault detection module are defined.

RULE_101 represents the knowledge required to detect a fault. It first checks to see if the ToolBook Graphical User Interface (GUI) has requested fault detection. Then it scans the *READING* properties of all objects in the *SENSORS* class. It will fire if it finds any sensors who's reading is *"BAD."* Then, it executes an externally defined handler which communicates this result to the GUI.

Code Segment 6.1: Rules for the Detection of a Fault

```
(@VERSION=      020)

(@OBJECT=       A_Fault_Has_Been_Detected
        (@PROPERTIES=  Value     @TYPE=Boolean;  )          )

(@OBJECT=       A_Fault_Has_Not_Been_Detected
        (@PROPERTIES=  Value     @TYPE=Boolean;  )          )

(@OBJECT=       Transponder_Functioning_Properly
        (@PROPERTIES=  Value     @TYPE=Boolean;  )          )


(@RULE=         RULE_101__DETECTION_OF_A_FAULT
        (@LHS= (Yes      (TBK_Request.Detection))
                         (Is       (<|SENSORS|>.READING)  ("BAD")  )          )
        (@HYPO=          A_Fault_Has_Been_Detected)
        (@RHS= (Execute  ("FaultDetected"))  )          )

(@RULE=         RULE_102__NON_DETECTION_OF_A_FAULT
        (@LHS= (Yes      (TBK_Request.Detection))
                         (Is       ({|SENSORS|}.READING)    ("GOOD")          ))
        (@HYPO=          A_Fault_Has_Not_Been_Detected)
        (@RHS= (Execute  ("NoFaultDetected"))          )          )
```

When NEXPERT™ sends the message *"FaultDetected"* to the ToolBook™ GUI, a handler defined there takes control from the NEXPERT™ application and the inference process is suspended. The user is informed that a fault has been detected within the transponder system and why this conclusion was reached. The user then has two options. He may either choose to continue the diagnostic process or stop and enter new data.

If the user decides not to pursue the rest of the diagnostic process, the fault detection module is unloaded. The user is then returned to the sensor data input mode of the GUI. Should the user choose to continue with the rest of the diagnostic process, the GUI will then load and initiate the module for the next task, the fault isolation module. This module is discussed in the next chapter.

RULE_101 can only report results to the ToolBook™ GUI if a fault has been detected. This is because the right hand side actions of a rule will only be executed when the rule fires. Furthermore, in atom-based inferencing, as used by NEXPERT™, there is no *"ELSE"* structure. This presents a problem in reporting to the GUI that the fault detection module has failed to detect a fault; implying that the transponder system is functioning properly. The result is the requirement of an additional rule.

RULE_102 represents the knowledge required to determine that the transponder is functioning properly. It first checks to see if the ToolBook Graphical User Interface (GUI) has requested fault detection. Then it scans the *READING* properties of all objects in the *SENSORS* class. It will fire only if it finds that all sensors are reporting *"GOOD."* Then, it executes an externally defined handler which communicates this result to the GUI.

The GUI handler for this message informs the user that the data he has entered is consistent with the proper functioning of the transponder system. The inference process is terminated, the fault detection module is unloaded, and the fidex kernel knowledge base is reset. The user then has the options to either end his session or evaluate a new set of data.

CHAPTER VII
FAULT ISOLATION MODULE KNOWLEDGE BASE

This chapter discusses the rule module for fault isolation knowledge. Recall from chapter 1 that each task in the diagnostic process was separated into an individual knowledge base. The knowledge for the isolation of faults is contained in the ISOLATE.tkb knowledge base. A complete listing of that module can be found in Appendix C.

In section 1.3, the purpose of the fault isolation task was defined as to isolate a suspected fault to a subsystem of the transponder. It was explained how this task also involved the analysis of current sensor information in the form of qualitative descriptions for sensor readings. The knowledge required to isolate a fault was then summarized as finding a subsystem who's input sensor is reporting a *"GOOD"* reading and who's output sensor is reporting a *"BAD"* reading.

Section 1.3 also discusses a subtask of sensor validation. This task was designed to identify the possibility of a faulty sensor. The knowledge required to validate sensor readings is the converse of that for isolating a fault. It can be summarizes as finding a subsystem who's input sensor is reporting a *"BAD"* reading and who's output sensor is reporting a *"GOOD"* reading. The following sections discuss the representation of knowledge required for each of these tasks.

7.1 Isolation of Faults to a Transponder Subsystem

The knowledge required to isolate a fault is defined in Code Segment 7.1. First, the objects which represent the hypotheses of rules are defined. Then the two rules of the fault isolation task are defined.

Code Segment 7.1: Rules for the Isolation of a Fault

```
(@VERSION=      020)

(@OBJECT=       Isolate_Fault_Symptoms
        (@PROPERTIES=  Value     @TYPE=Boolean;  )          )

(@RULE=         RULE_201__ISOLATION_OF_FAULT_TO_SUBSYSTEM
        (@LHS= (Yes        (TBK_Request.Isolation))
                           (Yes      (Model_Matrix_Switch_SubSystems)      )
                           (Is       (<|SUBSYSTEMS|>.READING_IN)    ("GOOD"))
                           (Is       (<|SUBSYSTEMS|>.READING_OUT) ("BAD")))
        (@HYPO=            Isolate_fault_Symptoms)
        (@RHS= (Let        (<|SUBSYSTEMS|>.ISOLATED)      (TRUE)  )
                           (CreateObject        (<|SUBSYSTEMS|>)               \
                           (|ISOLATED_SUB_SYSTEMS)
                           (Execute  ("FaultIsolated"))    )          )

(@RULE=         RULE_202__ISOLATION_OF_FAULT_TO_FREQUENCY_COMPONENTS
        (@LHS= (Yes        (TBK_Request.Isolation))
                           (Yes      (Model_Matrix_Switch_SubSystems)        )
                           (NotMember         ({|BAD_SENSORS|})                \
                                              (<|PWR_SENSORS|>)        )
                           (Is       (<|BER_SENSORS|>.READING)        ("BAD")))
        (@HYPO=            Isolate_fault_Symptoms)
        (@RHS= (CreateObject        (FREQ_COMPONENTS)                          \
                           (|ISOLATED_SUB_SYSTEMS)
                           (Let      (FREQ_COMPONENTS.ISOLATED)     (TRUE)  )
                           (Execute  ("FaultIsolated"))    )          )
```

RULE_201 represents the knowledge required to isolate a fault to one of the subsystems of the transponder. It first checks to see if the ToolBook Graphical User Interface (GUI) has requested fault isolation. Next, it verifies that the matrix switch paths have been modeled as subsystems of the transponder. Then it scans the *READING_IN* and *READING_OUT* properties of all objects in the *SUBSYSTEMS* class. It will fire if it finds any subsystem who's input reading is *"GOOD"* and output reading is *"BAD."* If the rule fires, it flags the subsystem which met its conditions as *ISOLATED* and creates a list of *ISOLATED_SUB_SYSTEMS* in case more than one fault exists within the transponder. And finally, it executes an externally defined handler in the GUI.

For this case, the purpose for sending this message to the GUI is simply to suspend the inferencing process and turn control over to the GUI. Recall from section 5.5.2 that slot dynamics associated with the *ISOLATED* property of *SUBSYSTEMS* effect the loading and initialization of subsystem diagnostic modules.

RULE_202 represents the knowledge required to isolate a fault to the group of frequency components in the transponder. It also first checks to see if the ToolBook Graphical User Interface (GUI) has requested fault isolation and that the matrix switch paths have been modeled as subsystems of the transponder. However, this rule scans the list of bad sensors that was created during the evaluation of sensor readings. It checks for the condition where no signal power level sensors report *"BAD"* readings, and at least one bit error rate register does. If it fires, it dynamically creates an object called *FREQ_COMPONENTS* and attaches it to the *ISOLATED_SUB_SYSTEMS* class. This new object will inherit subsystem properties, and its *ISOLATED* flag is set to *TRUE*. And finally, as for *RULE_201*, it executes an externally defined handler which communicates these results to the GUI

7.2 Validation of Sensorory Information

The knowledge required to validate sensor readings is defined in Code Segment 7.3. First, an object to represent the hypothesis of the rule is defined. Then the single rule of the sensor validation task is defined. *RULE_203* represents the knowledge required to determine that the detected fault may be the result of a faulty sensor. Like those for fault isolation, it first checks to see if the ToolBook Graphical User Interface (GUI) has requested fault isolation and that the matrix switch paths have been modeled as subsystems of the transponder.

Code Segment 7.2: Rules for the Validation of Sensors

```
(@OBJECT=        Validate_Sensors
        (@PROPERTIES=  Value      @TYPE=Boolean;  )          )

(@RULE=          RULE_203__VALIDATE_SENSOR_DATA
        (@LHS= (Yes      (TBK_Request.Isolation))
                         (Yes      (Model_Matrix_Switch_SubSystems)       )
                         (Is       (<|SUBSYSTEMS|>.READING_IN)   ("BAD"))
                         (Is       (<|SUBSYSTEMS|>.READING_OUT) ("GOOD")))
        (@HYPO=          Validate_Sensors)
        (@RHS= (CreateObject       (SENSOR_COMPONENTS)                       \
                         (|ISOLATED_SUB_SYSTEMS|)
                         (Let      (SENSOR_COMPONENTS.ISOLATED)  (TRUE)   )
                         (Execute  ("SensorFault"))      )          )
```

However, this rule scans the *READING_IN* and *READING_OUT* properties of all objects in the *SUBSYSTEMS* class for the condition of any subsystem who's input reading is *"BAD"* and output reading is *"GOOD."* If it fires, it dynamically creates an object called *SENSOR_COMPONENTS* and attaches it to the *ISOLATED_SUB_SYSTEMS* class. This new object will inherit subsystem properties, and its *ISOLATED* flag is set to *TRUE*. And finally, it executes an externally defined handler which communicates these results to the GUI.

The handler for the *"SensorFault"* takes control and the NEXPERT inference process is suspended. The user is informed that the data he has entered is not consistent with the behavior of a fault in the transponder. He is also told that it is possible that this is the result of a faulty sensor; an invalid sensor reading. The user then has the options to reenter the sensory information, or to continue with the diagnostic process. Should he choose to continue, the remainder of the diagnostic process is pursued as discussed in section 1.3. Recall from that section the special conditions that will be considered should a sensor fault be suspected.

CHAPTER VIII
FAULT DIAGNOSIS AND RESPONSE KNOWLEDGE BASES

This chapter discusses the community of specialized diagnostic expert systems for the fault diagnosis and response tasks in the diagnostic process. Recall from chapter 1 that each task in the diagnostic process was separated into an individual knowledge base. Furthermore, the rule knowledge required to diagnose faults associated with each subsystem of the transponder were also separated into specialized diagnostic knowledge bases. Listings for these knowledge bases can be found in Appendices D through G.

These specialized diagnostic systems use knowledge which is rule-based and backward chaining in nature. The hypotheses for these rules represent the potential faults in the isolated subsystem. The order in which they are placed on the agenda is based on the history of the fault states. Maintaining this history permits FIDEX to pursue the most likely problems first.

Each diagnostic system was also designed with an ability to perform inexact reasoning. This was done to overcome problems which resulted from limited information about the transponder's performance. Such an ability was important in that the FIDEX system would often need to make a "guess" at the most likely fault state. This relies upon establishing incremental measures of belief or disbelief in rule conclusions. These two factors are then used to establish an overall confidence when a conclusion is supported by multiple rules.

The final task of fault response has been incorporated within the diagnostic modules for each subsystem. The present strategy for fault response is to provide recommendations for reconfiguring the components or sensors. Plans are to include the capability to reconsider fault diagnosis if the recommended action was ineffective. FIDEX would retain its past diagnosis, including recommendations, and reconsider the problem with information made available following the corrections to the transponder.

There are several databases used by FIDEX. Seven of these are used to provide FIDEX a limited learning capability. FIDEX stores the failure history of the transponder subsystems system in associated database. Each known fault state is represented by a record that contains fields that represent the failure history of that fault state. Following diagnostics, FIDEX increments the history of the identified fault. This record keeping is used to direct the search strategy of future sessions toward the most likely faults. The next section discusses how this technique was implemented in all the diagnostic knowledgebases.

8.1 Learning and Adaptive Search Strategies

When NEXPERT's inference mechanism needs to process a slot or rule hypothesis, it does so according to their inference priorities. This processing occurs by the order of highest-first. The value of a slot or hypothesis's inference priority may be initialized from its default value of 1 to any integer between -32,000 and 32,000.

Inference priorities can be dynamically changed to allow slots to be processed with different priorities at different times. This is Neuron Data's mechanism for allowing the application to adapt itself to changing conditions. To implement this, a special slot called an *inference slot* is assigned to a slot's inference priority. The inference priority then takes on the value contained by the inference slot. This value is called the *inference number* of the atom.

The search strategy is adaptive in that the priorities by which known fault states are placed on the agenda is based upon the values maintained in the history database. A class level property of all fault states is the integer *INF_CAT*. The value of this property is retrieved from the database when the diagnostic task is initialized. This property is then assigned to the value of the inference slot of the fault state hypothesis. When the diagnostic task establishes a known fault state, the value of its inference category is incremented accordingly. The updated value is then stored in the learning database.

Recall the definition of *INF_CAT* property from section 4.4. This integer property is shared by all objects in the *FAULT_STATES* class. As each object in that class represents a known fault state, this property is linked as the inference slot of fault state hypotheses. Code Segment 4.1 shows this linking for the *AMP_GENERAL_FAILURE* (Amplifier General Failure) fault state.

Code Segment 8.1: Linking of *FAULT_STATE* Inference Slots

```
(@SLOT=        AMP_GENERAL_FAILURE.VERIFIED
        @INFATOM=   AMP_GENERAL_FAILURE.INF_CAT;      )
```

One limitation of the NEXPERT inference mechanism is that if the value of a slot defined as an inference slot is *UNKNOWN*, NEXPERT will not try to evaluate a value for that slot. Therefore, the values of *<|FAULT_STATES|>.INF_CAT* must be retrieved from the database before the diagnostic session begins. This initialization is effected by placing a rule on the agenda and forcing its evaluation. Code Segment 8.2 gives the definition of this rule.

Code Segment 8.2: Retrieval of <|*FAULT_STATES*|>.*INF_CAT*

```
(@RULE=         RULE_397_RETRIEVE_INFERENCE_CATEGORIES_FROM_DATABASE
        (@LHS=  (Yes        (Initialize_INF_CATs) )
        (@HYPO=             Retrieve_INF_CATs_From_DataBase)
        (@RHS=  (Retrieve   ("CH1RCVR.nxp")                                  \
                (@TYPE= NXPDB;                                               \
                 @FWRD= FALSE;                                               \
                 @UNKNOW= TRUE;                                              \
                 @NAME= "<|FAULT_STATES|>"                                   \
                 @PROPS= INF_CAT;                                            \
                 @FIELDS= "INF_CAT";)       )         )          )
```

During the initialization of a diagnostic module (ie. the channel 1 receiver subsystem diagnostic module in Code Segment 8.2), a value of *TRUE* is volunteered for *Initialize_INF_CATs.* This places *RULE_397* on the agenda and causes it to fire. The right hand side action then retrieves all inference category values form the designated database. In short, this discussion has addressed the recalling of inference categories from previous sessions with FIDEX. The final topic of this section is to address how FIDEX stores, or learns, about new inference categories for its fault state hypotheses.

During the initialization of diagnostic knowledge bases, the IC actions associated with slots are disabled, *@CACTIONSON= FALSE;* . Once the initialization process has completed, they are enabled again, *@CACTIONSON= TRUE;* . This allows changing the value of a fault state's *INF_CAT* slot to trigger the saving of a new inference category.

Code Segment 8.3: Storing <|*FAULT_STATES*|>.*INF_CAT*

```
(@SLOT= FAULT_STATES.INF_CAT
        (@CACTIONS=     (Write      ("CH1RCVR.nxp")                          \
                        (@TYPE= NXPDB;                                       \
                         @FWRD= FALSE;                                       \
                         @UNKNOW= TRUE;                                      \
                         @PROPS= INF_CAT;                                    \
                         @FIELDS= "INF_CAT";                                 \
                         @ATOMS= SELF;)  )          )
```

Code Segment 8.3 shows how changing the value of any *FAULT_STATES* inference category will result in that new value being written to the learning database. The values for the inference category are incremented by rules which support the fault state hypotheses. Code Segment 8.4 gives an example of such a rule.

Code Segment 8.4: Incrementing <|*FAULT_STATES*|>.*INF_CAT*

```
(@RULE=           RULE_301__ATTENUATOR_SETTING_ERROR
        (@LHS=  (>         (IFPC_ATTN_1.SETTING_ERROR)          (2.0))
        (@HYPO=            ATTENUATOR_SETTING_ERROR.VERIFIED)
        (@RHS=  (Do        (ATTENUATOR_SETTING_ERROR.INF_CAT+1)                  \
                           (ATTENUATOR_SETTING_ERROR.INF_CAT))         )         )
```

Code Segment 8.4 shows a rule that supports a fault state hypothesis that an IFPC attenuator has been set wrong. If this rule fires, its action will add 1 to the value of the inference category for the fault state *ATTENUATOR_SETTING_ERROR*.

8.2 Subsystems Diagnostic Modules

The operation of all the diagnostic modules is identical. Therefore, they will be discussed together. Complete listings may be found in the appendices. The following discussion will be in a general sense for all fault states.

Again, each known fault state is represented by an object attached to the class of *FAULT_STATES* in the object space of the diagnostic expert systems. After initialization, a suggestion is made that all fault states are to be verified, *@SUGGESTLST=* <|*FAULT_STATES*|>.*VERIFIED*. This suggestion places all fault states on the agenda. This is, in fact, an ordered list. According to the discussion of the previous section, the order by which they are placed on the agenda will be based on the values of their *INF_CAT* properties.

The inference strategy used in the diagnostic process to this point has been forward chaining, or data driven. Now, the strategy for diagnosing fault states turns to backward chaining. Each fault state hypothesis is taken from the agenda, in turn, and pursued exhaustivly. Associative rule knowledge which supports each hypothesis will be evaluated in an attempt to conclude the existence of a predefined fault state.

As the symptoms for these hard coded failure modes are pursued, evidence is also accumulated toward belief in, or rejection of, cumulatively associative and abstract fault states. Once the entire list has been evaluated, the fault state with the highest confidence factor is then presented as the diagnosed fault. The mechanisms of this inexact reasoning were discusses in several previous chapters.

Again, as discussed in previous chapters, it is possible that the entire list be evaluated and no fault state indicate sufficient certainty. It is here where the accumulation of evidence at the abstract, or class, level becomes significant. FIDEX is able to recover from its failure to diagnose the fault state by presenting a classification of faults as the diagnosed fault. This is to say, FIDEX has the capacity to offer conclusions of the form: *"The observed symptoms do not correspond to any known fault states of the transponder system. However, they do appear to be consistent with those of an amplifier failure."*

Another important aspect of the architecture of the fault diagnostic modules is the facility with which diagnostic knowledge may be maintained. As the experience of the SITE personnel grows, and more knowledge is gained about the failure modes of the transponder, this knowledge can be appended to the knowledge bases by two simple manners. First, should more knowledge be gained about an existing fault, new rules may be added. By attaching these new rules to the existing fault state hypotheses, they will automatically be incorporated into the inference strategy of the diagnostic module.

Should a new fault state be discovered, the procedure is slightly more involved, but not more complex. A new fault state can be added to a diagnostic module by first creating an object to represent it and attaching that object into the *FAULT_STATES* hierarchy. Then as discussed earlier, any rule knowledge which supports that new fault state can be encoded into the inference strategy by attaching it to the new hypothesis.

CHAPTER IX
SUMMARY OF RESEARCH

FIDEX, the Fault Isolation and Diagnosis Expert System, was the result of a research contract with the National Aeronautics and Space Administration (NASA), Lewis Research Center, in Cleveland Ohio. It was designed to provide intelligent computer diagnostics for a Ka-band satellite transponder system, as part of the Advanced Communication Technology Satellite (ACTS) System. The overall goal of this research was to enhance the reliability of the satellite by demonstrating the application of expert system technologies as a means for providing the transponder with an autonomous diagnosis capability.

The results of this research were more than just the development of a prototype diagnosis expert system. The frame-based approach that was taken proved that hierarchical structures are the best means for representing complex structures such as the satellite's: subsystems, components, sensors, and fault states. Furthermore, FIDEX demonstrated that integrating these hierarchical structures into a lattice provided a flexible representation scheme that greatly facilitated the maintenance of the system architecture.

FIDEX's ability to detect abnormalities in the sensors which provide information on the transponder's performance proved effective for ruling out simple sensor malfunctions. However, The major strengths of the FIDEX system have appeared on two different fronts.

First, FIDEX proved that inexact reasoning techniques, based on the incrementally acquired evidence, are effective means for overcoming limitations on the availability of information. This approach enabled FIDEX to reason in an abstract sense, and thus recover from situations where no concrete diagnostic conclusion could be reached. Furthermore, the frame-based architecture which resulted for the implementation of these techniques greatly facilitates the matter of maintaining the knowledge which supports the diagnosis of fault states.

The second major strength of the FIDEX system was that it demonstrated that a primitive databased learning ability was an effective approach to maintaining records of past diagnosis studies. This ability permitted FIDEX to adapt its diagnostic search strategies to search first for those faults which are most likely to occur. Moreover, as most diagnostic expert systems learn through adaptations of new diagnostic knowledge, FIDEX enhanced its intelligence by learning about the diagnostic process itself.

In its present form, FIDEX is a prototype system that is practical for demonstration purposes only. However, its overall design, with its generic structures and innovative features, makes the FIDEX system an applicable example for other types of diagnostic systems. Furthermore, the stress placed on the maintainability of its architecture provides for future developments and expansions which encompass a wide range of possibilities. Primarily however, it is the hope of its developers that the FIDEX system will eventually be augmented into a fully autonomous diagnostic expert system.

REFERENCES

[1] **Bagwell, J.W.**, *"A System for the Simulation and Evaluation of Satellite Communication Networks,"* 10th AIAA Communication Satellite Systems Conference, Proceedings of, January 1984.

[2] **Barr, A., and Feigenbaum E.A., (eds.)**, The Handbook of Artificial Intelligence, vol. 1, William Kaufman, 1981.

[3] **Bobrow, D.G., etal**, *"CommonLoops: Merging Lisp and Object-Oriented Programming,"* The Conference on Object-Oriented Programming Systems, Languages and Applications, Proceedings of, 1986, pp.17-29.

[4] **Boehm, B.W.**, Characteristics of Software Quality, Elsevier North-Holland (New York, NY), 1978.

[5] **Buzzard, G.D., and Mudge, T.N.**, *"Object-Based Computing and the Ada Programming Language,"* Computer, March 1985, pp.11-19.

[6] **Cox, B.J.**, *"Message/Object Programming: An Evolutionary Change in Programming Technology,"* IEEE Software, January 1984, pp.50-61.

[7] **Diederick, J., and Milton, J.**, *"Experimental Prototyping in Smalltalk,"* IEEE Software, May 1987, pp.50-64.

[8] **Durkin, J.**, Expert System Design and Development, Macmillan Publishing Co.(New York, NY), 1993.

[9] **Durkin, J., Tallo, D.P., and Petrik E.**, *"FIDEX: An Expert System for Satellite Diagnostics,"* Second Space Communications Technology Conference: Onboard Processing and Switching, NASA S&T Information Division Publication 3132, pp.143-52, Proceedings of Conference at NASA-Lewis Research Center, (Cleveland, OH), November 12-14, 1991.

[10] **Dutta, K.**, *"Modular Programming in C: An Approach and an Example,"* SIGPLAN Notices, March 1985, pp.9-15.

[11] **Erman, L.D.**, *"The HEARSAY-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainties,"* Computing Surveys, Vol.12, No.2, June 1980, pp.213-253.

[12] **Fujikawa, G., and Kerczewski, R.J.**, *"Performance of a Ka-Band Satellite System Under Variable Signal Power Conditions,"* 1987 IEEE MTT Microwave Symposium, Proceedings of, June 1987.

[13] **Harmon, P., Maus, R., and Morrissey, W.**, Expert Systems Tools and Applications, John Wiley and Sons (New York, NY), 1988.

[14] **Ingalls, D.H.H.**, *"Design Principles Behind Smalltalk,"* Byte, August 1981, pp.286-298.

[15] **Kay, A.C.**, *"Microelectronics and the Personal Computer,"* Scientific American, September 1977, pp.231-244.

[16] **Kerczewski, R.J., and Fujikawa, G.**, *"Performance Measurements for a Laboratory-Simulated 30/20 GHz Communication Satellite Transponder,"* 13th AIAA International Communication Satellite Conference, Proceedings of, March 1990.

[17] **Leonard, R.F., and Kerczewski, R.J.**, *"Radiofrequency Testing of Satellite Segment of Simulated 30/20 GHz Satellite Communication Systems,"* Internal Publication, NASA-Lewis Research Center (Cleveland, OH), November 1985.

[18] **Love, T.**, *"Experiences with Smalltalk-80™ for Application Development,"* Softfair: A Conference on Software Development Tools, Techniques, and Alternatives, Proceedings of, 1983, pp.61-65.

[19] **Ledbetter, L., Cox, B.**, *"Software-IC's,"* Byte, June 1985, pp.307-315.

[20] **MacLennan, B.J.**, *"Values and Objects in Programming Languages,"* SIGPLAN Notices, December 1982, pp.70-79.

[21] **Minsky, M.L.**, *"Frame System Theory,"* Thinking: Readings in Cognitive Science, ed. P.N. Johnson-Laird and P.C. Watson, Cambridge (CUP), 1975.

[22] **Pascoe, G.A.**, *"Elements of Object-Oriented Programming,"* Byte, August 1986, pp.307-316.

[23] **Peterson, G.E.**, *"Introduction to Object-Oriented Computing,"* TUTORIAL: Object-Oriented Computing, IEEE Computer Society Press (Los Alamitos, CA) 1990, pp.1-4.

[24] **Peterson, G.E.**, *"Smalltalk: An Object-Based Language,"* TUTORIAL: Object-Oriented Computing, IEEE Computer Society Press (Los Alamitos, CA) 1990, pp.37-40.

[25] **Peterson, G.E.**, *"Object-Oriented Programming in Ada,"* TUTORIAL: Object-Oriented Computing, IEEE Computer Society Press (Los Alamitos, CA) 1990, pp.97-98.

[26] **Peterson, G.E.**, *"Object-Oriented Programming Languages,"* TUTORIAL: Object-Oriented Computing, IEEE Computer Society Press (Los Alamitos, CA) 1990, pp.133-136.

[27] **Pfleeger, S.L.**, Software Engineering-The Production of Quality Software, Macmillan Publishing Co. (New York, NY), 1987.

[28] **Reenskaug, T.M.H.**, *"User-Oriented Descriptions of Smalltalk Systems,"* Byte, August 1981, pp.148-166.

[29] **Rentsch, T.**, *"Object-Oriented Programming,"* SIGPLAN Notices, September 1982, pp. 51-57.

[30] **Robson, D.**, *"Object-Oriented Software Systems,"* Byte, August 1981, pp. 74-86.

[31] **Schlegelmilch, R., Durkin, J., and Petrik, E.**, *"GTEX: An Expert System for Diagnosing Faults in Satellite Ground Stations,"* Second Space Communications Technology Conference: Onboard Processing and Switching, NASA S&T Information Division Publication 3132, pp.103-112, Proceedings of Conference at NASA-Lewis Research Center, (Cleveland, OH), November 12-14, 1991.

[32] **Schmucker, K.J.**, *"Object-Oriented Languages for the Macintosh,"* Byte, August 1986, pp.177-185.

[33] **Shalkhauser, K.A., and Kerczewski, R.J.**, *"Automated Testing of Satellite Communications Systems and Subsystems,"* 25th Automatic RF Techniques Group Conference, Proceedings of, June 1985.

[34] **Shortliffe, E.H. and Buchanan, B.G.**, *"A model of Inexact Reasoning in Medicine,"* Mathematical Biosciences, Vol.23, 1975.

[35] **Stefik, M., and Bobrow, D.G.**, *"Object-Oriented Programming: Themes and Variations,"* The AI Magazine, Winter Edition 1986, pp.40-62.

[36] **Stephan, A., and Erikson, C.A.**, *"GETTING EXPERT SYSTEMS OFF THE GROUND: Lessons Learned from Integrating Model-based Diagnostics with Prototype Flight Hardware,"* Second Space Communications Technology Conference: Onboard Processing and Switching, NASA S&T Information Division Publication 3132, pp.135-42, Proceedings of Conference at NASA-Lewis Research Center, (Cleveland, OH), November 12-14, 1991.

[37] **Tallo, D.P., Durkin, J., and Petrik, E.**, *"Intelligent Fault Isolation and Diagnosis for Communication Satellite Systems,"* 1992 Goddard Conference on Space Applications of Artificial Intelligence, NASA S&T Information Division Publication TBA, pp.TBA, Proceedings of Conference at NASA-Goddard Space Flight Center, (Greenbelt, MD), May 5-7, 1992.

[38] **Volz, R.A., Mudge, T.N., and Gal, D.A.**, *"Using Ada as a Programming Language for Robot-Based Manufacturing Cells,"* IEEE Transactions on Systems, Man, and Cybernetics, November/December 1984, pp. 863-878.

[39] **Waterman, D.A.**, A Guide to Expert Systems, Addison-Wesley (Reading, MS), 1986.

[40] **Webster**, Webster's New World Dictionary, Second College Edition, William Collins World Publishing Co., Inc., (Cleveland, OH) 1976.

[41] **Wegner, P.**, *"On the Unification of Data and Program Abstraction in Ada,"* 10th Annual ACM Symposium on Principles of Programming Languages, Proceedings of, 1983, pp. 256-264.

[42] **Windmiller, M.J.**, *"Unique Bit-Error-Rate Measurement System for Satellite Communication Systems,"* Internal Publication, NASA-Lewis Research Center (Cleveland, OH), March 1987.

[43] **Wirth, N.**, *"History and Goals of Modula-2,"* Byte, August 1984, pp.145-152.

[44] **Xerox Learning Research Group, The**, *"The Smalltalk-80 System,"* Byte, August 1981, pp.36-48.

APPENDIX A
FIDEX KERNEL KNOWLEDGE BASE

A.1 Sensor Initialization Database

```
                         Name¦        NOMINAL¦      TOLERANCE¦      ZERO_LEVEL¦
********************************************************************************
                        BER_1¦            0.0¦          0.005¦             0.0¦
                        BER_2¦            0.0¦          0.005¦             0.0¦
                        BER_3¦            0.0¦          0.005¦             0.0¦
                        BER_4¦            0.0¦          0.005¦             0.0¦
                        BER_5¦            0.0¦          0.005¦             0.0¦
                        BER_6¦            0.0¦          0.005¦             0.0¦
                         PM_1¦           -6.0¦            2.0¦           -30.0¦
                         PM_2¦           -6.0¦            2.0¦           -30.0¦
                         PM_3¦            0.0¦            2.0¦           -30.0¦
                         PM_4¦            0.0¦            2.0¦           -30.0¦
                         PM_5¦          -15.0¦            2.0¦           -30.0¦
                         PM_6¦          -15.0¦            2.0¦           -30.0¦
                         PM_7¦            5.0¦            2.0¦           -30.0¦
                         PM_8¦            5.0¦            2.0¦           -30.0¦
********************************************************************************
```

A.2 Components Initialization Database

Name	COMPONENT_IN	COMPONENT_OUT	NASA_ID	NOM_FREQ	NOM_FREQ_IN	NOM_FREQ_OUT	NOM_GAIN	NOM_POWER_IN	NOM_POWER_OUT
GAASFET	HPAPC_ATTN_1	NONE		20.0	20.0	20.0	35.0	-5.0	30.0
HPAPC_AMP_1	HPAPC_ATTN_2	HPAPC_ATTN_1		5.0	5.0	5.0	NotKnown	NotKnown	NotKnown
HPAPC_AMP_2	HPAPC_ATTN_3	HPAPC_ATTN_4		5.0	5.0	5.0	NotKnown	NotKnown	NotKnown
HPAPC_ATTN_1	HPAPC_AMP_1	GAASFET	RA17	5.0	5.0	5.0	NotKnown	NotKnown	-5.0
HPAPC_ATTN_2	MULT_1	HPAPC_AMP_1	MA16	5.0	5.0	5.0	NotKnown	NotKnown	NotKnown
HPAPC_ATTN_3	MULT_2	HPAPC_AMP_2	MA26	5.0	5.0	5.0	NotKnown	NotKnown	NotKnown
HPAPC_ATTN_4	HPAPC_AMP_2	TWTA	RA16	5.0	5.0	5.0	NotKnown	NotKnown	-5.0
IFPC_AMP_1	RCVR_1	IFPC_ATTN_1		5.0	5.0	5.0	43.0	-22.0	21.0
IFPC_AMP_2	RCVR_2	IFPC_ATTN_2		5.0	5.0	5.0	43.0	-22.0	21.0
IFPC_AMP_3	MSWITCH	IFPC_ATTN_3		5.0	5.0	5.0	45.0	-28.0	17.0
IFPC_AMP_4	MSWITCH	IFPC_ATTN_4		5.0	5.0	5.0	45.0	-28.0	17.0
IFPC_ATTN_1	IFPC_AMP_1	MSWITCH	SA12	5.0	5.0	5.0	-17.0	21.0	4.0
IFPC_ATTN_2	IFPC_AMP_2	MSWITCH	SA22	5.0	5.0	5.0	-17.0	21.0	4.0
IFPC_ATTN_3	IFPC_AMP_3	MULT_1	SA14	5.0	5.0	5.0	-7.0	17.0	10.0
IFPC_ATTN_4	IFPC_AMP_4	MULT_2	SA24	5.0	5.0	5.0	-7.0	17.0	10.0
MSWITCH	IFPC_ATTN_1	IFPC_AMP_3		5.0	5.0	5.0	-32.0	4.0	-28.0
MULT_1	IFPC_ATTN_3	HPAPC_ATTN_2		14.2	5.0	20.0	NotKnown	10.0	NotKnown
MULT_2	IFPC_ATTN_4	HPAPC_ATTN_3		14.2	5.0	20.0	NotKnown	10.0	NotKnown
RCVR_1	NONE	IFPC_AMP_1		23.8	30.0	5.0	NotKnown	NotKnown	-22.0
RCVR_2	NONE	IFPC_AMP_2		23.8	30.0	5.0	NotKnown	NotKnown	-22.0
RCVR_LO	NONE	RCVR_1		23.8	0.0	23.8	12.0	0.0	12.0
TWTA	HPAPC_ATTN_4	NONE		20.0	20.0	20.0	NotKnown	NotKnown	NotKnown
UPX_LO	NONE	MULT_1		14.2	0.0	14.2	9.0	0.0	9.0

A.3 FIDEX Kernel Knowledge Base

```
(@VERSION=   020)
(@PROPERTY=  AB  @TYPE=Float;)
(@PROPERTY=  AD  @TYPE=Float;)
(@PROPERTY=  BAD @TYPE=Boolean;)
(@PROPERTY=  Bad_Sensors     @TYPE=Boolean;)
(@PROPERTY=  BIAS_CURRENT     @TYPE=Float;)
(@PROPERTY=  BIAS_VOLTAGE     @TYPE=Float;)
(@PROPERTY=  CF  @TYPE=Float;)
(@PROPERTY=  COMPONENT    @TYPE=String;)
(@PROPERTY=  COMPONENT_IN     @TYPE=String;)
(@PROPERTY=  COMPONENT_IN_2   @TYPE=String;)
(@PROPERTY=  COMPONENT_OUT    @TYPE=String;)
(@PROPERTY=  COMPONENT_OUT_2 @TYPE=String;)
(@PROPERTY=  CONFIDENCE  @TYPE=String;)
(@PROPERTY=  CONFIG  @TYPE=String;)
(@PROPERTY=  COUPLING     @TYPE=Boolean;)
(@PROPERTY=  DATA    @TYPE=Float;)
(@PROPERTY=  DESCRIPTION @TYPE=String;)
(@PROPERTY=  DIAGNOSTIC_MODULE   @TYPE=Boolean;)
(@PROPERTY=  DRAIN_VOLTAGE   @TYPE=Float;)
(@PROPERTY=  ERROR   @TYPE=Float;)
(@PROPERTY=  EVALUATED   TYPE=Boolean;)
(@PROPERTY=  FREQUENCY   @TYPE=Float;)
(@PROPERTY=  FREQUENCY_2 @TYPE=Float;)
(@PROPERTY=  FREQUENCY_IN     @TYPE=Float;)
(@PROPERTY=  FREQUENCY_IN_2   @TYPE=Float;)
(@PROPERTY=  FREQUENCY_OUT    @TYPE=Float;)
(@PROPERTY=  FREQUENCY_OUT_2 @TYPE=Float;)
(@PROPERTY=  GAIN    @TYPE=Float;)
(@PROPERTY=  GAIN_2  @TYPE=Float;)
(@PROPERTY=  GATE_VOLTAGE     @TYPE=Float;)
(@PROPERTY=  GOOD    @TYPE=Boolean;)
(@PROPERTY=  HIGH    @TYPE=Boolean;)
(@PROPERTY=  INF_CAT @TYPE=Float;)
(@PROPERTY=  ISOLATED    @TYPE=Boolean;)
(@PROPERTY=  LEVEL   @TYPE=String;)
(@PROPERTY=  LEVEL_IN    @TYPE=String;)
(@PROPERTY=  LEVEL_OUT   @TYPE=String;)
(@PROPERTY=  LO_INPUT_FREQUENCY  @TYPE=Float;)
(@PROPERTY=  LO_INPUT_POWER  @TYPE=Float;)
(@PROPERTY=  LO_UNIT @TYPE=String;)
(@PROPERTY=  LOW @TYPE=Boolean;)
(@PROPERTY=  MB  @TYPE=Float;)
(@PROPERTY=  MD  @TYPE=Float;)
(@PROPERTY=  MODEL_GAIN  @TYPE=Float;)
(@PROPERTY=  MODEL_GAIN_2    @TYPE=Float;)
(@PROPERTY=  MODEL_POWER_IN  @TYPE=Float;)
(@PROPERTY=  MODEL_POWER_IN_2    @TYPE=Float;)
(@PROPERTY=  MODEL_POWER_OUT @TYPE=Float;)
(@PROPERTY=  MODEL_POWER_OUT_2   @TYPE=Float;)
(@PROPERTY=  MODEL_SETTING   @TYPE=Float;)
(@PROPERTY=  NAME    @TYPE=String;)
(@PROPERTY=  NASA_ID @TYPE=String;)
(@PROPERTY=  NOMINAL @TYPE=Float;)
(@PROPERTY=  NOMINAL_BIAS_CURRENT     @TYPE=Float;)
(@PROPERTY=  NOMINAL_BIAS_VOLTAGE     @TYPE=Float;)
(@PROPERTY=  NOMINAL_DRAIN_VOLTAGE    @TYPE=Float;)
(@PROPERTY=  NOMINAL_FREQUENCY    @TYPE=Float;)
(@PROPERTY=  NOMINAL_FREQUENCY_2  @TYPE=Float;)
(@PROPERTY=  NOMINAL_FREQUENCY_IN     @TYPE=Float;)
(@PROPERTY=  NOMINAL_FREQUENCY_IN_2   @TYPE=Float;)
(@PROPERTY=  NOMINAL_FREQUENCY_OUT    @TYPE=Float;)
(@PROPERTY=  NOMINAL_FREQUENCY_OUT_2  @TYPE=Float;)
(@PROPERTY=  NOMINAL_GAIN    @TYPE=Float;)
(@PROPERTY=  NOMINAL_GATE_VOLTAGE     @TYPE=Float;)
(@PROPERTY=  NOMINAL_LO_INPUT_FREQUENCY@TYPE=Float;)
(@PROPERTY=  NOMINAL_LO_INPUT_POWER@TYPE=Float;)
```

```
(@PROPERTY=  NOMINAL_POWER_IN     @TYPE=Float;)
(@PROPERTY=  NOMINAL_POWER_IN_2   @TYPE=Float;)
(@PROPERTY=  NOMINAL_POWER_OUT    @TYPE=Float;)
(@PROPERTY=  NOMINAL_POWER_OUT_2 @TYPE=Float;)
(@PROPERTY=  Nominal_Sensor_Data @TYPE=Boolean;)
(@PROPERTY=  NOMINAL_SETTING @TYPE=Float;)
(@PROPERTY=  OK  @TYPE=Boolean;)
(@PROPERTY=  POWER_IN    @TYPE=Float;)
(@PROPERTY=  POWER_IN_2  @TYPE=Float;)
(@PROPERTY=  POWER_OUT   @TYPE=Float;)
(@PROPERTY=  POWER_OUT_2 @TYPE=Float;)
(@PROPERTY=  READING @TYPE=String;)
(@PROPERTY=  READING_IN  @TYPE=String;)
(@PROPERTY=  READING_OUT @TYPE=String;)
(@PROPERTY=  RTN_LEVEL   @TYPE=Boolean;)
(@PROPERTY=  RTN_NOMINAL @TYPE=Boolean;)
(@PROPERTY=  RTN_READING @TYPE=Boolean;)
(@PROPERTY=  SENSOR_IN   @TYPE=String;)
(@PROPERTY=  SENSOR_OUT  @TYPE=String;)
(@PROPERTY=  SETTING @TYPE=Float;)
(@PROPERTY=  SETTING_ERROR   @TYPE=Float;)
(@PROPERTY=  TOLERANCE   @TYPE=Float;)
(@PROPERTY=  TYPE    @TYPE=String;)
(@PROPERTY=  VERIFIED    @TYPE=Boolean;)
(@PROPERTY=  ZERO    @TYPE=Boolean;)
(@PROPERTY=  ZERO_LEVEL  @TYPE=Float;)
```

```
(@CLASS=    AMPLIFIER_FAULTS
    (@PROPERTIES=
        AB
        AD
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MD
        NAME
        VERIFIED
    )
)

(@CLASS=    AMPLIFIERS
    (@PROPERTIES=
        BIAS_CURRENT
        BIAS_VOLTAGE
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        DRAIN_VOLTAGE
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        GATE_VOLTAGE
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL_BIAS_CURRENT
        NOMINAL_BIAS_VOLTAGE
        NOMINAL_DRAIN_VOLTAGE
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_GATE_VOLTAGE
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
    )
)

(@CLASS=    ATTENUATOR_FAULTS
    (@PROPERTIES=
        AB
        AD
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MD
        NAME
        VERIFIED
    )
)

(@CLASS=    ATTENUATORS
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        MODEL_SETTING
        NAME
        NASA_ID
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        NOMINAL_SETTING
        POWER_IN
        POWER_OUT
        SETTING
        SETTING_ERROR
    )
)

(@CLASS=    BAD_SENSORS
    (@PROPERTIES=
        RTN_READING
    )
)

(@CLASS=    BER_REGISTERS
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
    )
)

(@CLASS=    BER_SENSORS
    (@SUBCLASSES=
        CH1_BERs
        CH2_BERs
    )
    (@PROPERTIES=
        DATA
        ERROR
        EVALUATED
        LEVEL
        NAME
        NOMINAL
        READING
        RTN_LEVEL
        RTN_NOMINAL
        RTN_READING
        TOLERANCE
        TYPE
        ZERO_LEVEL
    )
)
```

```
(@CLASS=    CERTAINTY_ANALYSIS
    (@SUBCLASSES=
        FAULT_STATES
    )
    (@PROPERTIES=
        AB
        AD
        CF
        CONFIDENCE
        MB
        MD
    )
)

(@CLASS=    CH1_BERs
    (@PROPERTIES=
        DATA
        ERROR
        EVALUATED
        LEVEL
        NAME
        NOMINAL
        READING
        RTN_LEVEL
        RTN_NOMINAL
        RTN_READING
        TOLERANCE
        TYPE
        ZERO_LEVEL
    )
)

(@CLASS=    CH2_BERs
    (@PROPERTIES=
        DATA
        ERROR
        EVALUATED
        LEVEL
        NAME
        NOMINAL
        READING
        RTN_LEVEL
        RTN_NOMINAL
        RTN_READING
        TOLERANCE
        TYPE
        ZERO_LEVEL
    )
)

(@CLASS=    COMPONENTS
    (@SUBCLASSES=
        AMPLIFIERS
        ATTENUATORS
        LOCAL_OSCILLATORS
        RECEIVERS
        POWER_METERS
        BER_REGISTERS
        SWITCHES
        GaAsFETS
        TWTAS
        MIXERS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
    )
)

(@CLASS=    FAULT_STATES
    (@SUBCLASSES=
        AMPLIFIER_FAULTS
        ATTENUATOR_FAULTS
        GaAs_FET_FAULTS
        LO_FAULTS
        MIXER_FAULTS
        RECEIVER_FAULTS
        SWITCH_FAULTS
        TWTA_FAULTS
    )
    (@PROPERTIES=
        AB
        AD
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MD
        NAME
        VERIFIED
    )
)

(@CLASS=    GaAs_FET_FAULTS
    (@PROPERTIES=
        AB
        AD
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MD
        NAME
        VERIFIED
    )
)

(@CLASS=    GaAsFETS
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        DRAIN_VOLTAGE
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        GATE_VOLTAGE
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL_DRAIN_VOLTAGE
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
```

```
            NOMINAL_FREQUENCY_OUT
            NOMINAL_GAIN
            NOMINAL_GATE_VOLTAGE
            NOMINAL_POWER_IN
            NOMINAL_POWER_OUT
            POWER_IN
            POWER_OUT
        )
)

(@CLASS=     LO_FAULTS
    (@PROPERTIES=
            AB
            AD
            CF
            COMPONENT
            CONFIDENCE
            INF_CAT
            MB
            MD
            NAME
            VERIFIED
        )
)

(@CLASS=     LOCAL_OSCILLATORS
    (@PROPERTIES=
            COMPONENT_IN
            COMPONENT_OUT
            COMPONENT_OUT_2
            DESCRIPTION
            FREQUENCY
            FREQUENCY_IN
            FREQUENCY_OUT
            FREQUENCY_OUT_2
            GAIN
            MODEL_GAIN
            MODEL_POWER_IN
            MODEL_POWER_OUT
            NAME
            NASA_ID
            NOMINAL_FREQUENCY
            NOMINAL_FREQUENCY_IN
            NOMINAL_FREQUENCY_OUT
            NOMINAL_FREQUENCY_OUT_2
            NOMINAL_GAIN
            NOMINAL_POWER_IN
            NOMINAL_POWER_OUT
            NOMINAL_POWER_OUT_2
            POWER_IN
            POWER_OUT
            POWER_OUT_2
        )
)

(@CLASS=     MIXER_FAULTS
    (@PROPERTIES=
            AB
            AD
            CF
            COMPONENT
            CONFIDENCE
            INF_CAT
            MB
            MD
            NAME
            VERIFIED
        )
)

(@CLASS=     MIXERS
    (@PROPERTIES=
            COMPONENT_IN
            COMPONENT_OUT
            DESCRIPTION
            FREQUENCY
            FREQUENCY_IN
            FREQUENCY_OUT
            GAIN
            LO_INPUT_FREQUENCY
            LO_INPUT_POWER
            LO_UNIT
            MODEL_GAIN
            MODEL_POWER_IN
            MODEL_POWER_OUT
            NAME
            NASA_ID
            NOMINAL_FREQUENCY
            NOMINAL_FREQUENCY_IN
            NOMINAL_FREQUENCY_OUT
            NOMINAL_GAIN
            NOMINAL_LO_INPUT_FREQUENCY
            NOMINAL_LO_INPUT_POWER
            NOMINAL_POWER_IN
            NOMINAL_POWER_OUT
            POWER_IN
            POWER_OUT
        )
)

(@CLASS=     POWER_METERS
    (@PROPERTIES=
            COMPONENT_IN
            COMPONENT_OUT
            DESCRIPTION
            FREQUENCY
            FREQUENCY_IN
            FREQUENCY_OUT
            GAIN
            MODEL_GAIN
            MODEL_POWER_IN
            MODEL_POWER_OUT
            NAME
            NASA_ID
            NOMINAL_FREQUENCY
            NOMINAL_FREQUENCY_IN
            NOMINAL_FREQUENCY_OUT
            NOMINAL_GAIN
            NOMINAL_POWER_IN
            NOMINAL_POWER_OUT
            POWER_IN
            POWER_OUT
        )
)

(@CLASS=     PWR_SENSORS
    (@PROPERTIES=
            DATA
            ERROR
            EVALUATED
            LEVEL
            NAME
            NOMINAL
            READING
            RTN_LEVEL
            RTN_NOMINAL
            RTN_READING
            TOLERANCE
            TYPE
            ZERO_LEVEL
        )
)
```

```
(@CLASS=     RECEIVER_FAULTS
    (@PROPERTIES=
        AB
        AD
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MD
        NAME
        VERIFIED
    )
)

(@CLASS=     RECEIVERS
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        LO_INPUT_FREQUENCY
        LO_INPUT_POWER
        LO_UNIT
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_LO_INPUT_FREQUENCY
        NOMINAL_LO_INPUT_POWER
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
    )
)

(@CLASS=     SENSORS
    (@SUBCLASSES=
        PWR_SENSORS
        BER_SENSORS
    )
    (@PROPERTIES=
        DATA
        ERROR
        EVALUATED
        LEVEL
        NAME
        NOMINAL
        READING
        RTN_LEVEL
        RTN_NOMINAL
        RTN_READING
        TOLERANCE
        TYPE
        ZERO_LEVEL
    )
)

(@CLASS=     SUBSYSTEMS
    (@PROPERTIES=
        DIAGNOSTIC_MODULE
        ISOLATED
        LEVEL_IN
        LEVEL_OUT
        NAME
        READING_IN
        READING_OUT
        SENSOR_IN
        SENSOR_OUT
    )
)

(@CLASS=     SWITCH_FAULTS
    (@PROPERTIES=
        AB
        AD
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MD
        NAME
        VERIFIED
    )
)

(@CLASS=     SWITCHES
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_IN_2
        COMPONENT_OUT
        COMPONENT_OUT_2
        DESCRIPTION
        FREQUENCY
        FREQUENCY_2
        FREQUENCY_IN
        FREQUENCY_IN_2
        FREQUENCY_OUT
        FREQUENCY_OUT_2
        GAIN
        GAIN_2
        MODEL_GAIN
        MODEL_GAIN_2
        MODEL_POWER_IN
        MODEL_POWER_IN_2
        MODEL_POWER_OUT
        MODEL_POWER_OUT_2
        NAME
        NASA_ID
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_2
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_IN_2
        NOMINAL_FREQUENCY_OUT
        NOMINAL_FREQUENCY_OUT_2
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_IN_2
        NOMINAL_POWER_OUT
        NOMINAL_POWER_OUT_2
        POWER_IN
        POWER_IN_2
        POWER_OUT
        POWER_OUT_2
    )
)

(@CLASS=     TWTA_FAULTS
    (@PROPERTIES=
        AB
        AD
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MD
        NAME
        VERIFIED
    )
)

(@CLASS=    TWTAS
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
    )
)

(@OBJECT=   BER_1
    (@CLASSES=
        CH1_BERs
        BER_REGISTERS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DATA
        DESCRIPTION
        ERROR
        EVALUATED
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        LEVEL
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
        READING
        RTN_LEVEL
        RTN_NOMINAL
        RTN_READING
        TOLERANCE
        TYPE
        ZERO_LEVEL
    )
)

(@OBJECT=   BER_2
    (@CLASSES=
        CH1_BERs
        BER_REGISTERS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DATA
        DESCRIPTION
        ERROR
        EVALUATED
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        LEVEL
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
        READING
        RTN_LEVEL
        RTN_NOMINAL
        RTN_READING
        TOLERANCE
        TYPE
        ZERO_LEVEL
    )
)
```

```
(@OBJECT=   BER_3
    (@CLASSES=
        CH1_BERs
        BER_REGISTERS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DATA
        DESCRIPTION
        ERROR
        EVALUATED
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        LEVEL
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
        READING
        RTN_LEVEL
        RTN_NOMINAL
        RTN_READING
        TOLERANCE
        TYPE
        ZERO_LEVEL
    )
)

(@OBJECT=   BER_4
    (@CLASSES=
        CH2_BERs
        BER_REGISTERS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DATA
        DESCRIPTION
        ERROR
        EVALUATED
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        LEVEL
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
        READING
        RTN_LEVEL
        RTN_NOMINAL
        RTN_READING
        TOLERANCE
        TYPE
        ZERO_LEVEL
    )
)

(@OBJECT=   BER_5
    (@CLASSES=
        CH2_BERs
        BER_REGISTERS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DATA
        DESCRIPTION
        ERROR
        EVALUATED
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        LEVEL
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
        READING
        RTN_LEVEL
        RTN_NOMINAL
        RTN_READING
        TOLERANCE
        TYPE
        ZERO_LEVEL
    )
)

(@OBJECT=   BER_6
    (@CLASSES=
        CH2_BERs
        BER_REGISTERS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DATA
        DESCRIPTION
        ERROR
        EVALUATED
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        LEVEL
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
        READING
        RTN_LEVEL
        RTN_NOMINAL
        RTN_READING
        TOLERANCE
        TYPE
        ZERO_LEVEL
    )
)

(@OBJECT=   CH1AMP
    (@CLASSES=
        SUBSYSTEMS
    )
    (@PROPERTIES=
        DIAGNOSTIC_MODULE
        ISOLATED
        LEVEL_IN
        LEVEL_OUT
        NAME
        READING_IN
        READING_OUT
        SENSOR_IN
        SENSOR_OUT
    )
)

(@OBJECT=   CH1RCVR
    (@CLASSES=
        SUBSYSTEMS
    )
    (@PROPERTIES=
        DIAGNOSTIC_MODULE
        ISOLATED
        LEVEL_IN
        LEVEL_OUT
        NAME
        READING_IN
        READING_OUT
        SENSOR_IN
        SENSOR_OUT
    )
)

(@OBJECT=   CH1UPX
    (@CLASSES=
        SUBSYSTEMS
    )
    (@PROPERTIES=
        DIAGNOSTIC_MODULE
        ISOLATED
        LEVEL_IN
        LEVEL_OUT
        NAME
        READING_IN
        READING_OUT
        SENSOR_IN
        SENSOR_OUT
    )
)

(@OBJECT=   CH2AMP
    (@CLASSES=
        SUBSYSTEMS
    )
    (@PROPERTIES=
        DIAGNOSTIC_MODULE
        ISOLATED
        LEVEL_IN
        LEVEL_OUT
        NAME
        READING_IN
        READING_OUT
        SENSOR_IN
        SENSOR_OUT
    )
)

(@OBJECT=   CH2RCVR
    (@CLASSES=
        SUBSYSTEMS
    )
    (@PROPERTIES=
        DIAGNOSTIC_MODULE
        ISOLATED
        LEVEL_IN
        LEVEL_OUT
        NAME
        READING_IN
        READING_OUT
        SENSOR_IN
        SENSOR_OUT
    )
)

(@OBJECT=   CH2UPX
    (@CLASSES=
        SUBSYSTEMS
    )
    (@PROPERTIES=
        DIAGNOSTIC_MODULE
        ISOLATED
        LEVEL_IN
        LEVEL_OUT
        NAME
        READING_IN
        READING_OUT
        SENSOR_IN
        SENSOR_OUT
    )
)

(@OBJECT=   CURRENT_COMPONENT
    (@PROPERTIES=
        COUPLING
        NAME
    )
)

(@OBJECT=   CURRENT_FAULT
    (@PROPERTIES=
        NAME
    )
)

(@OBJECT=   CURRENT_SENSOR
    (@PROPERTIES=
        NAME
    )
)

(@OBJECT=   CURRENT_SUBSYSTEM
    (@PROPERTIES=
        LEVEL_IN
        LEVEL_OUT
        NAME
        READING_IN
        READING_OUT
        SENSOR_IN
        SENSOR_OUT
    )
)

(@OBJECT=   Evaluate_Certainty_Factors
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@OBJECT=   GAASFET
    (@CLASSES=
        GaAsFETS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        DRAIN_VOLTAGE
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        GATE_VOLTAGE
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL_DRAIN_VOLTAGE
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_GATE_VOLTAGE
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
    )
)

(@OBJECT=   HPAPC_AMP_1
    (@CLASSES=
        AMPLIFIERS
    )
    (@PROPERTIES=
        BIAS_CURRENT
        BIAS_VOLTAGE
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        DRAIN_VOLTAGE
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        GATE_VOLTAGE
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL_BIAS_CURRENT
        NOMINAL_BIAS_VOLTAGE
        NOMINAL_DRAIN_VOLTAGE
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_GATE_VOLTAGE
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
    )
)

(@OBJECT=   HPAPC_AMP_2
    (@CLASSES=
        AMPLIFIERS
    )
    (@PROPERTIES=
        BIAS_CURRENT
        BIAS_VOLTAGE
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        DRAIN_VOLTAGE
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        GATE_VOLTAGE
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL_BIAS_CURRENT
        NOMINAL_BIAS_VOLTAGE
        NOMINAL_DRAIN_VOLTAGE
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_GATE_VOLTAGE
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
    )
)

(@OBJECT=   HPAPC_ATTN_1
    (@CLASSES=
        ATTENUATORS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        MODEL_SETTING
        NAME
        NASA_ID
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        NOMINAL_SETTING
        POWER_IN
        POWER_OUT
        SETTING
        SETTING_ERROR
    )
)

(@OBJECT=   HPAPC_ATTN_2
    (@CLASSES=
        ATTENUATORS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        MODEL_SETTING
        NAME
        NASA_ID
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        NOMINAL_SETTING
        POWER_IN
        POWER_OUT
        SETTING
        SETTING_ERROR
    )
)

(@OBJECT=   HPAPC_ATTN_3
    (@CLASSES=
        ATTENUATORS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        MODEL_SETTING
        NAME
        NASA_ID
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        NOMINAL_SETTING
        POWER_IN
        POWER_OUT
        SETTING
        SETTING_ERROR
    )
)

(@OBJECT=   HPAPC_ATTN_4
    (@CLASSES=
        ATTENUATORS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        MODEL_SETTING
        NAME
        NASA_ID
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        NOMINAL_SETTING
        POWER_IN
        POWER_OUT
        SETTING
        SETTING_ERROR
    )
)

(@OBJECT=   IFPC_AMP_1
    (@CLASSES=
        AMPLIFIERS
    )
    (@PROPERTIES=
        BIAS_CURRENT
        BIAS_VOLTAGE
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        DRAIN_VOLTAGE
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        GATE_VOLTAGE
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL_BIAS_CURRENT
        NOMINAL_BIAS_VOLTAGE
        NOMINAL_DRAIN_VOLTAGE
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_GATE_VOLTAGE
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
    )
)
```

```
(@OBJECT=    IFPC_AMP_2
    (@CLASSES=
        AMPLIFIERS
    )
    (@PROPERTIES=
        BIAS_CURRENT
        BIAS_VOLTAGE
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        DRAIN_VOLTAGE
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        GATE_VOLTAGE
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL_BIAS_CURRENT
        NOMINAL_BIAS_VOLTAGE
        NOMINAL_DRAIN_VOLTAGE
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_GATE_VOLTAGE
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
    )
)

(@OBJECT=    IFPC_AMP_3
    (@CLASSES=
        AMPLIFIERS
    )
    (@PROPERTIES=
        BIAS_CURRENT
        BIAS_VOLTAGE
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        DRAIN_VOLTAGE
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        GATE_VOLTAGE
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL_BIAS_CURRENT
        NOMINAL_BIAS_VOLTAGE
        NOMINAL_DRAIN_VOLTAGE
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_GATE_VOLTAGE
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
    )
)

(@OBJECT=    IFPC_AMP_4
    (@CLASSES=
        AMPLIFIERS
    )
    (@PROPERTIES=
        BIAS_CURRENT
        BIAS_VOLTAGE
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        DRAIN_VOLTAGE
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        GATE_VOLTAGE
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL_BIAS_CURRENT
        NOMINAL_BIAS_VOLTAGE
        NOMINAL_DRAIN_VOLTAGE
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_GATE_VOLTAGE
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
    )
)

(@OBJECT=    IFPC_ATTN_1
    (@CLASSES=
        ATTENUATORS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        MODEL_SETTING
        NAME
        NASA_ID
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        NOMINAL_SETTING
        POWER_IN
        POWER_OUT
        SETTING
        SETTING_ERROR
    )
)
```

```
(@OBJECT=   IFPC_ATTN_2
    (@CLASSES=
        ATTENUATORS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        MODEL_SETTING
        NAME
        NASA_ID
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        NOMINAL_SETTING
        POWER_IN
        POWER_OUT
        SETTING
        SETTING_ERROR
    )
)

(@OBJECT=   IFPC_ATTN_3
    (@CLASSES=
        ATTENUATORS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        MODEL_SETTING
        NAME
        NASA_ID
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        NOMINAL_SETTING
        POWER_IN
        POWER_OUT
        SETTING
        SETTING_ERROR
    )
)

(@OBJECT=   IFPC_ATTN_4
    (@CLASSES=
        ATTENUATORS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        MODEL_SETTING
        NAME
        NASA_ID
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        NOMINAL_SETTING
        POWER_IN
        POWER_OUT
        SETTING
        SETTING_ERROR
    )
)

(@OBJECT=   Model_Matrix_Switch_SubSystem
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@OBJECT=   MSWITCH
    (@CLASSES=
        SWITCHES
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_IN_2
        COMPONENT_OUT
        COMPONENT_OUT_2
        CONFIG
        DESCRIPTION
        FREQUENCY
        FREQUENCY_2
        FREQUENCY_IN
        FREQUENCY_IN_2
        FREQUENCY_OUT
        FREQUENCY_OUT_2
        GAIN
        GAIN_2
        MODEL_GAIN
        MODEL_GAIN_2
        MODEL_POWER_IN
        MODEL_POWER_IN_2
        MODEL_POWER_OUT
        MODEL_POWER_OUT_2
        NAME
        NASA_ID
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_2
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_IN_2
        NOMINAL_FREQUENCY_OUT
        NOMINAL_FREQUENCY_OUT_2
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_IN_2
        NOMINAL_POWER_OUT
        NOMINAL_POWER_OUT_2
        POWER_IN
        POWER_IN_2
        POWER_OUT
        POWER_OUT_2
    )
)
```

```
(@OBJECT=    MSWITCH_CH11
    (@PROPERTIES=
        DIAGNOSTIC_MODULE
        ISOLATED
        LEVEL_IN
        LEVEL_OUT
        NAME
        READING_IN
        READING_OUT
        SENSOR_IN
        SENSOR_OUT
    )
)

(@OBJECT=    MSWITCH_CH12
    (@PROPERTIES=
        DIAGNOSTIC_MODULE
        ISOLATED
        LEVEL_IN
        LEVEL_OUT
        NAME
        READING_IN
        READING_OUT
        SENSOR_IN
        SENSOR_OUT
    )
)

(@OBJECT=    MSWITCH_CH21
    (@PROPERTIES=
        DIAGNOSTIC_MODULE
        ISOLATED
        LEVEL_IN
        LEVEL_OUT
        NAME
        READING_IN
        READING_OUT
        SENSOR_IN
        SENSOR_OUT
    )
)

(@OBJECT=    MSWITCH_CH22
    (@PROPERTIES=
        DIAGNOSTIC_MODULE
        ISOLATED
        LEVEL_IN
        LEVEL_OUT
        NAME
        READING_IN
        READING_OUT
        SENSOR_IN
        SENSOR_OUT
    )
)

(@OBJECT=    MULT_1
    (@CLASSES=
        MIXERS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        LO_INPUT_FREQUENCY
        LO_INPUT_POWER
        LO_UNIT
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_LO_INPUT_FREQUENCY
        NOMINAL_LO_INPUT_POWER
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
    )
)

(@OBJECT=    MULT_2
    (@CLASSES=
        MIXERS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        LO_INPUT_FREQUENCY
        LO_INPUT_POWER
        LO_UNIT
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_LO_INPUT_FREQUENCY
        NOMINAL_LO_INPUT_POWER
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
    )
)

(@OBJECT=    PM_0
    (@PROPERTIES=
        LEVEL
        NAME
        READING
    )
)

(@OBJECT=    PM_1
    (@CLASSES=
        POWER_METERS
        PWR_SENSORS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DATA
        DESCRIPTION
        ERROR
        EVALUATED
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        LEVEL
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
        READING
        RTN_LEVEL
        RTN_NOMINAL
        RTN_READING
        TOLERANCE
        TYPE
        ZERO_LEVEL
    )
)

(@OBJECT=   PM_2
    (@CLASSES=
        POWER_METERS
        PWR_SENSORS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DATA
        DESCRIPTION
        ERROR
        EVALUATED
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        LEVEL
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
        READING
        RTN_LEVEL
        RTN_NOMINAL
        RTN_READING
        TOLERANCE
        TYPE
        ZERO_LEVEL
    )
)

(@OBJECT=   PM_3
    (@CLASSES=
        POWER_METERS
        PWR_SENSORS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DATA
        DESCRIPTION
        ERROR
        EVALUATED
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        LEVEL
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
        READING
        RTN_LEVEL
        RTN_NOMINAL
        RTN_READING
        TOLERANCE
        TYPE
        ZERO_LEVEL
    )
)

(@OBJECT=   PM_4
    (@CLASSES=
        POWER_METERS
        PWR_SENSORS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DATA
        DESCRIPTION
        ERROR
        EVALUATED
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        LEVEL
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
        READING
        RTN_LEVEL
        RTN_NOMINAL
        RTN_READING
        TOLERANCE
        TYPE
        ZERO_LEVEL
    ))
```

```
(@OBJECT=   PM_5
    (@CLASSES=
        POWER_METERS
        PWR_SENSORS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DATA
        DESCRIPTION
        ERROR
        EVALUATED
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        LEVEL
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
        READING
        RTN_LEVEL
        RTN_NOMINAL
        RTN_READING
        TOLERANCE
        TYPE
        ZERO_LEVEL
    )
)

(@OBJECT=   PM_6
    (@CLASSES=
        POWER_METERS
        PWR_SENSORS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DATA
        DESCRIPTION
        ERROR
        EVALUATED
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        LEVEL
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
        READING
        RTN_LEVEL
        RTN_NOMINAL
        RTN_READING
        TOLERANCE
        TYPE
        ZERO_LEVEL
    )
)

(@OBJECT=   PM_7
    (@CLASSES=
        POWER_METERS
        PWR_SENSORS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DATA
        DESCRIPTION
        ERROR
        EVALUATED
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        LEVEL
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
        READING
        RTN_LEVEL
        RTN_NOMINAL
        RTN_READING
        TOLERANCE
        TYPE
        ZERO_LEVEL
    )
)

(@OBJECT=   PM_8
    (@CLASSES=
        POWER_METERS
        PWR_SENSORS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DATA
        DESCRIPTION
        ERROR
        EVALUATED
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        LEVEL
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
        READING
        RTN_LEVEL
        RTN_NOMINAL
        RTN_READING
        TOLERANCE
        TYPE
        ZERO_LEVEL
    )
)

(@OBJECT=   RCVR_1
    (@CLASSES=
        RECEIVERS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        LO_INPUT_FREQUENCY
        LO_INPUT_POWER
        LO_UNIT
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_LO_INPUT_FREQUENCY
        NOMINAL_LO_INPUT_POWER
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
    )
)

(@OBJECT=   RCVR_2
    (@CLASSES=
        RECEIVERS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        LO_INPUT_FREQUENCY
        LO_INPUT_POWER
        LO_UNIT
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_LO_INPUT_FREQUENCY
        NOMINAL_LO_INPUT_POWER
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
    )
)

(@OBJECT=   RCVR_LO
    (@CLASSES=
        LOCAL_OSCILLATORS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        COMPONENT_OUT_2
        DESCRIPTION
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        FREQUENCY_OUT_2
        GAIN
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_FREQUENCY_OUT_2
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        NOMINAL_POWER_OUT_2
        POWER_IN
        POWER_OUT
        POWER_OUT_2
    )
)

(@OBJECT=   Return_BAD_Sensors
    (@PROPERTIES=
        Value    @TYPE=Boolean;
    )
)

(@OBJECT=   Return_Nominal_Sensor_Data
    (@PROPERTIES=
        Value    @TYPE=Boolean;
    )
)

(@OBJECT=   Sensor_Level_Description
    (@PROPERTIES=
        HIGH
        LOW
        OK
        ZERO
    )
)

(@OBJECT=   Sensor_Reading_Description
    (@PROPERTIES=
        BAD
        GOOD
    )
)
```

```
(@OBJECT=   TBK_Request
    (@PROPERTIES=
        Bad_Sensors
        Nominal_Sensor_Data
    )
)

(@OBJECT=   TWTA
    (@CLASSES=
        TWTAS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        DESCRIPTION
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        GAIN
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        POWER_IN
        POWER_OUT
    )
)

(@OBJECT=   UPX_LO
    (@CLASSES=
        LOCAL_OSCILLATORS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        COMPONENT_OUT_2
        DESCRIPTION
        FREQUENCY
        FREQUENCY_IN
        FREQUENCY_OUT
        FREQUENCY_OUT_2
        GAIN
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NASA_ID
        NOMINAL_FREQUENCY
        NOMINAL_FREQUENCY_IN
        NOMINAL_FREQUENCY_OUT
        NOMINAL_FREQUENCY_OUT_2
        NOMINAL_GAIN
        NOMINAL_POWER_IN
        NOMINAL_POWER_OUT
        NOMINAL_POWER_OUT_2
        POWER_IN
        POWER_OUT
        POWER_OUT_2
    )
)
        (@SLOT= BER_SENSORS.TYPE
            (@INITVAL=   "BER")
            (@SOURCES=
                (RunTimeValue    ("BER"))
            )
        )
```

```
(@SLOT= CERTAINTY_ANALYSIS.AB
    (@INITVAL=  0.0)
    (@SOURCES=
        (RunTimeValue   (0.0))
    )
    (@CACTIONS=
        (Do ((SELF.AB-SELF.AD)/MIN(SELF.AB,SELF.AD))  (SELF.CF))
    )
)

(@SLOT= CERTAINTY_ANALYSIS.AD
    (@INITVAL=  0.0)
    (@SOURCES=
        (RunTimeValue   (0.0))
    )
    (@CACTIONS=
        (Do ((SELF.AB-SELF.AD)/MIN(SELF.AB,SELF.AD))  (SELF.CF))
    )
)

(@SLOT= CERTAINTY_ANALYSIS.CF
    (@INITVAL=  0.0)
    (@SOURCES=
        (RunTimeValue   (0.0))
    )
    (@CACTIONS=
        (Do (SELF.NAME)  (CURRENT_FAULT.NAME))
        (Reset  (Evaluate_Certainty_Factors))
        (Do (Evaluate_Certainty_Factors) (Evaluate_Certainty_Factors))
    )
)

(@SLOT= CERTAINTY_ANALYSIS.MB
    (@INITVAL=  0.0)
    (@SOURCES=
        (RunTimeValue   (0.0))
    )
    (@CACTIONS=
        (Do (SELF.AB+SELF.MB*(1-SELF.AB))    (SELF.AB))
        (Reset  (SELF.MB))
    )
)

(@SLOT= CERTAINTY_ANALYSIS.MD
    (@INITVAL=  0.0)
    (@SOURCES=
        (RunTimeValue   (0.0))
    )
    (@CACTIONS=
        (Do (SELF.AD+SELF.MD*(1-SELF.AD))    (SELF.AD))
        (Reset  (SELF.MD))
    )
)

(@SLOT= COMPONENTS.GAIN
    (@SOURCES=
        (Do (SELF.POWER_OUT-SELF.POWER_IN)   (SELF.GAIN))
    )
    (@CACTIONS=
        (Do (SELF.POWER_IN+SELF.GAIN)    (SELF.POWER_OUT))
    )
)

(@SLOT= COMPONENTS.MODEL_GAIN
    (@SOURCES=
        (Do (SELF.MODEL_POWER_OUT-SELF.MODEL_POWER_IN)    (SELF.MODEL_GAIN))
    )
    (@CACTIONS=
        (Do (SELF.MODEL_POWER_IN+SELF.MODEL_GAIN)    (SELF.MODEL_POWER_OUT))
    )
)
```

```
(@SLOT= COMPONENTS.MODEL_POWER_IN
    (@SOURCES=
        (Do (\SELF.COMPONENT_IN\.MODEL_POWER_OUT)    (SELF.MODEL_POWER_IN))
    )
    (@CACTIONS=
        (Do (SELF.MODEL_POWER_IN+SELF.MODEL_GAIN)    (SELF.MODEL_POWER_OUT))
    )
)

(@SLOT= COMPONENTS.MODEL_POWER_OUT
    (@SOURCES=
        (Do (SELF.MODEL_POWER_IN+SELF.MODEL_GAIN)    (SELF.MODEL_POWER_OUT))
    )
    (@CACTIONS=
        (Do (SELF.MODEL_POWER_OUT)   (\SELF.COMPONENT_OUT\.MODEL_POWER_IN))
    )
)

(@SLOT= COMPONENTS.POWER_IN
    (@SOURCES=
        (Do (\SELF.COMPONENT_IN\.POWER_OUT)  (SELF.POWER_IN))
    )
    (@CACTIONS=
        (Do (SELF.POWER_IN+SELF.GAIN)    (SELF.POWER_OUT))
    )
)

(@SLOT= COMPONENTS.POWER_OUT
    (@SOURCES=
        (Do (SELF.POWER_IN+SELF.GAIN)    (SELF.POWER_OUT))
    )
    (@CACTIONS=
        (Do (SELF.POWER_OUT)    (\SELF.COMPONENT_OUT\.POWER_IN))
    )
)

(@SLOT= PWR_SENSORS.TYPE
    (@INITVAL=  "PM")
    (@SOURCES=
        (RunTimeValue   ("PM"))
    )
)

(@SLOT= SENSORS.DATA
    (@CACTIONS=
        (Reset  (SELF.ERROR))
        (Reset  (SELF.READING))
        (Reset  (SELF.LEVEL))
        (Do (SELF.ERROR)    (SELF.ERROR))
        (Do (SELF.READING)  (SELF.READING))
        (Do (SELF.LEVEL)    (SELF.LEVEL))
    )
)

(@SLOT= SENSORS.ERROR
    (@SOURCES=
        (Do (SELF.DATA-SELF.NOMINAL) (SELF.ERROR))
    )
)

(@SLOT= SENSORS.LEVEL
    (@SOURCES=
        (Do (SELF.NAME) (CURRENT_SENSOR.NAME))
        (Reset  (Sensor_Level_Description.ZERO))
        (Do (Sensor_Level_Description.ZERO)  (Sensor_Level_Description.ZERO))
        (Reset  (Sensor_Level_Description.LOW))
        (Do (Sensor_Level_Description.LOW)   (Sensor_Level_Description.LOW))
        (Reset  (Sensor_Level_Description.HIGH))
        (Do (Sensor_Level_Description.HIGH)  (Sensor_Level_Description.HIGH))
        (Reset  (Sensor_Level_Description.OK))
        (Do (Sensor_Level_Description.OK)    (Sensor_Level_Description.OK))
    )
    (@CACTIONS=
        (Execute     ("ReturnSensorLevel")    (@ATOMID=SELF;@STRING="@V(@SELF.LEVEL)";\
))
    )
)

(@SLOT= SENSORS.NAME
    (@SOURCES=
        (Retrieve    ("SENSOR.nxp")   (@TYPE=NXPDB;@FWRD=FALSE;@UNKNOWN=TRUE;@PROPS=NAME,\
NOMINAL,TOLERANCE,ZERO_LEVEL;@FIELDS="NAME",\
"NOMINAL","TOLERANCE","ZERO_LEVEL";@ATOMS=SELF;\
))
    )
)

(@SLOT= SENSORS.NOMINAL
    (@SOURCES=
        (Retrieve    ("SENSOR.nxp")   (@TYPE=NXPDB;@FWRD=FALSE;@UNKNOWN=TRUE;@PROPS=NAME,\
NOMINAL,TOLERANCE,ZERO_LEVEL;@FIELDS="NAME",\
"NOMINAL","TOLERANCE","ZERO_LEVEL";@ATOMS=SELF;\
))
    )
    (@CACTIONS=
        (Execute     ("ReturnNominalData")    (@ATOMID=SELF;@STRING="@V(@SELF.NOMINAL)";\
))
    )
)

(@SLOT= SENSORS.READING
    (@SOURCES=
        (Do (SELF.NAME) (CURRENT_SENSOR.NAME))
        (Reset  (Sensor_Reading_Description.BAD))
        (Do (Sensor_Reading_Description.BAD) (Sensor_Reading_Description.BAD))
        (Reset  (Sensor_Reading_Description.GOOD))
        (Do (Sensor_Reading_Description.GOOD)    (Sensor_Reading_Description.GOOD))
    )
    (@CACTIONS=
        (Execute     ("ReturnSensorReading")  (@ATOMID=SELF;@STRING="@V(@SELF.READING)";\
))
    )
)

(@SLOT= SENSORS.RTN_LEVEL
    (@CACTIONS=
        (Execute     ("ReturnSensorLevel")    (@ATOMID=SELF;@STRING="@V(@SELF.LEVEL)";\
))
    )
)

(@SLOT= SENSORS.RTN_NOMINAL
    (@CACTIONS=
        (Execute     ("ReturnNominalData")    (@ATOMID=SELF;@STRING="@V(@SELF.NOMINAL)";\
))
    )
)

(@SLOT= SENSORS.RTN_READING
    (@CACTIONS=
        (Execute     ("ReturnSensorReading")  (@ATOMID=SELF;@STRING="@V(@SELF.READING)";\
))
    )
)

(@SLOT= SENSORS.TOLERANCE
    (@SOURCES=
        (Retrieve    ("SENSOR.nxp")   (@TYPE=NXPDB;@FWRD=FALSE;@UNKNOWN=TRUE;@PROPS=NAME,\
NOMINAL,TOLERANCE,ZERO_LEVEL;@FIELDS="NAME",\
"NOMINAL","TOLERANCE","ZERO_LEVEL";@ATOMS=SELF;\
))
    )
)
```

```
(@SLOT= SENSORS.ZERO_LEVEL
    (@SOURCES=
        (Retrieve   ("SENSOR.nxp")  (@TYPE=NXPDB;@FWRD=FALSE;@UNKNOWN=TRUE;@PROPS=NAME,\
NOMINAL,TOLERANCE,ZERO_LEVEL;@FIELDS="NAME",\
"NOMINAL","TOLERANCE","ZERO_LEVEL";@ATOMS=SELF;\
))
    )
)

(@SLOT= SUBSYSTEMS.ISOLATED
    (@CACTIONS=
        (Execute    ("ReturnIsolation")  (@ATOMID=SELF;@STRING="@V(@SELF.NAME)";))
    )
)

(@SLOT= SUBSYSTEMS.LEVEL_IN
    (@SOURCES=
        (Do (\SELF.SENSOR_IN\.LEVEL) (SELF.LEVEL_IN))
    )
)

(@SLOT= SUBSYSTEMS.LEVEL_OUT
    (@SOURCES=
        (Do (\SELF.SENSOR_OUT\.LEVEL)    (SELF.LEVEL_OUT))
    )
)

(@SLOT= SUBSYSTEMS.READING_IN
    (@SOURCES=
        (Do (\SELF.SENSOR_IN\.READING)   (SELF.READING_IN))
    )
)

(@SLOT= SUBSYSTEMS.READING_OUT
    (@SOURCES=
        (Do (\SELF.SENSOR_OUT\.READING)  (SELF.READING_OUT))
    )
)

(@SLOT= SWITCHES.GAIN_2
    (@SOURCES=
        (Do (SELF.POWER_OUT_2-SELF.POWER_IN_2)    (SELF.GAIN_2))
    )
    (@CACTIONS=
        (Do (SELF.POWER_IN_2+SELF.GAIN_2)    (SELF.POWER_OUT_2))
    )
)

(@SLOT= SWITCHES.MODEL_GAIN_2
    (@SOURCES=
        (Do (SELF.MODEL_POWER_OUT_2-SELF.MODEL_POWER_IN_2)    (SELF.MODEL_GAIN_2))
    )
    (@CACTIONS=
        (Do (SELF.MODEL_POWER_IN_2+SELF.MODEL_GAIN_2)    (SELF.MODEL_POWER_OUT_2))
    )
)

(@SLOT= BER_1.NAME
    (@INITVAL=  "BER_1")
    (@SOURCES=
        (RunTimeValue   ("BER_1"))
    )
)

(@SLOT= BER_2.NAME
    (@INITVAL=  "BER_2")
    (@SOURCES=
        (RunTimeValue   ("BER_2"))
    )
)
```

```
(@SLOT= BER_3.NAME
    (@INITVAL=  "BER_3")
    (@SOURCES=
        (RunTimeValue   ("BER_3"))
    )
)

(@SLOT= BER_4.NAME
    (@INITVAL=  "BER_4")
    (@SOURCES=
        (RunTimeValue   ("BER_4"))
    )
)

(@SLOT= BER_5.NAME
    (@INITVAL=  "BER_5")
    (@SOURCES=
        (RunTimeValue   ("BER_5"))
    )
)

(@SLOT= BER_6.NAME
    (@INITVAL=  "BER_6")
    (@SOURCES=
        (RunTimeValue   ("BER_6"))
    )
)

(@SLOT= CH1AMP.DIAGNOSTIC_MODULE
    (@INITVAL=  FALSE)
    (@SOURCES=
        (RunTimeValue   (FALSE))
    )
    (@CACTIONS=
        (LoadKB ("CH1AMP.tkb")  (@LEVEL=ENABLE;))
    )
)

(@SLOT= CH1AMP.NAME
    (@INITVAL=  "CH1AMP")
    (@SOURCES=
        (RunTimeValue   ("CH1AMP"))
    )
)

(@SLOT= CH1AMP.SENSOR_IN
    (@INITVAL=  "PM_5")
    (@SOURCES=
        (RunTimeValue   ("PM_5"))
    )
)

(@SLOT= CH1AMP.SENSOR_OUT
    (@INITVAL=  "PM_7")
    (@SOURCES=
        (RunTimeValue   ("PM_7"))
    )
)

(@SLOT= CH1RCVR.DIAGNOSTIC_MODULE
    (@INITVAL=  FALSE)
    (@SOURCES=
        (RunTimeValue   (FALSE))
    )
    (@CACTIONS=
        (LoadKB ("CH1RCVR.tkb") (@LEVEL=ENABLE;))
    )
)
```

```
(@SLOT= CH1RCVR.NAME
    (@INITVAL=  "CH1RCVR")
    (@SOURCES=
        (RunTimeValue   ("CH1RCVR"))
    )
)

(@SLOT= CH1RCVR.SENSOR_IN
    (@INITVAL=  "PM_0")
    (@SOURCES=
        (RunTimeValue   ("PM_0"))
    )
)

(@SLOT= CH1RCVR.SENSOR_OUT
    (@INITVAL=  "PM_1")
    (@SOURCES=
        (RunTimeValue   ("PM_1"))
    )
)

(@SLOT= CH1UPX.DIAGNOSTIC_MODULE
    (@INITVAL=  FALSE)
    (@SOURCES=
        (RunTimeValue   (FALSE))
    )
    (@CACTIONS=
        (LoadKB ("CH1UPX.tkb")  (@LEVEL=ENABLE;))
    )
)

(@SLOT= CH1UPX.NAME
    (@INITVAL=  "CH1UPX")
    (@SOURCES=
        (RunTimeValue   ("CH1UPX"))
    )
)

(@SLOT= CH1UPX.SENSOR_IN
    (@INITVAL=  "PM_3")
    (@SOURCES=
        (RunTimeValue   ("PM_3"))
    )
)

(@SLOT= CH1UPX.SENSOR_OUT
    (@INITVAL=  "PM_5")
    (@SOURCES=
        (RunTimeValue   ("PM_5"))
    )
)

(@SLOT= CH2AMP.DIAGNOSTIC_MODULE
    (@INITVAL=  FALSE)
    (@SOURCES=
        (RunTimeValue   (FALSE))
    )
    (@CACTIONS=
        (LoadKB ("CH2AMP.tkb")  (@LEVEL=ENABLE;))
    )
)

(@SLOT= CH2AMP.NAME
    (@INITVAL=  "CH2AMP")
    (@SOURCES=
        (RunTimeValue   ("CH2AMP"))
    )
)
```

```
(@SLOT= CH2AMP.SENSOR_IN
    (@INITVAL=  "PM_6")
    (@SOURCES=
        (RunTimeValue   ("PM_6"))
    )
)

(@SLOT= CH2AMP.SENSOR_OUT
    (@INITVAL=  "PM_8")
    (@SOURCES=
        (RunTimeValue   ("PM_8"))
    )
)

(@SLOT= CH2RCVR.DIAGNOSTIC_MODULE
    (@INITVAL=  FALSE)
    (@SOURCES=
        (RunTimeValue   (FALSE))
    )
    (@CACTIONS=
        (LoadKB ("CH2RCVR.tkb")  (@LEVEL=ENABLE;))
    )
)

(@SLOT= CH2RCVR.NAME
    (@INITVAL=  "CH2RCVR")
    (@SOURCES=
        (RunTimeValue   ("CH2RCVR"))
    )
)

(@SLOT= CH2RCVR.SENSOR_IN
    (@INITVAL=  "PM_0")
    (@SOURCES=
        (RunTimeValue   ("PM_0"))
    )
)

(@SLOT= CH2RCVR.SENSOR_OUT
    (@INITVAL=  "PM_2")
    (@SOURCES=
        (RunTimeValue   ("PM_2"))
    )
)

(@SLOT= CH2UPX.DIAGNOSTIC_MODULE
    (@INITVAL=  FALSE)
    (@SOURCES=
        (RunTimeValue   (FALSE))
    )
    (@CACTIONS=
        (LoadKB ("CH2UPX.tkb")   (@LEVEL=ENABLE;))
    )
)

(@SLOT= CH2UPX.NAME
    (@INITVAL=  "CH2UPX")
    (@SOURCES=
        (RunTimeValue   ("CH2UPX"))
    )
)

(@SLOT= CH2UPX.SENSOR_IN
    (@INITVAL=  "PM_4")
    (@SOURCES=
        (RunTimeValue   ("PM_4"))
    )
)
```

```
(@SLOT= CH2UPX.SENSOR_OUT
    (@INITVAL=  "PM_6")
    (@SOURCES=
        (RunTimeValue    ("PM_6"))
    )
)

(@SLOT= MSWITCH.CONFIG
    (@SOURCES=
        (Execute    ("RequestMatrixSwitchConfig"))
    )
)

(@SLOT= MSWITCH_CH11.DIAGNOSTIC_MODULE
    (@INITVAL=  FALSE)
    (@SOURCES=
        (RunTimeValue    (FALSE))
    )
    (@CACTIONS=
        (LoadKB  ("MSWITCH.tkb")  (@LEVEL=ENABLE;))
    )
)

(@SLOT= MSWITCH_CH11.NAME
    (@INITVAL=  "MSWITCH")
    (@SOURCES=
        (RunTimeValue    ("MSWITCH"))
    )
)

(@SLOT= MSWITCH_CH11.SENSOR_IN
    (@INITVAL=  "PM_1")
    (@SOURCES=
        (RunTimeValue    ("PM_1"))
    )
)

(@SLOT= MSWITCH_CH11.SENSOR_OUT
    (@INITVAL=  "PM_3")
    (@SOURCES=
        (RunTimeValue    ("PM_3"))
    )
)

(@SLOT= MSWITCH_CH12.DIAGNOSTIC_MODULE
    (@INITVAL=  FALSE)
    (@SOURCES=
        (RunTimeValue    (FALSE))
    )
    (@CACTIONS=
        (LoadKB  ("MSWITCH.tkb")  (@LEVEL=ENABLE;))
    )
)

(@SLOT= MSWITCH_CH12.NAME
    (@INITVAL=  "MSWITCH")
    (@SOURCES=
        (RunTimeValue    ("MSWITCH"))
    )
)

(@SLOT= MSWITCH_CH12.SENSOR_IN
    (@INITVAL=  "PM_1")
    (@SOURCES=
        (RunTimeValue    ("PM_1"))
    )
)

(@SLOT= MSWITCH_CH12.SENSOR_OUT
    (@INITVAL=  "PM_4")
    (@SOURCES=
        (RunTimeValue    ("PM_4"))
    ))
```

```
(@SLOT= MSWITCH_CH21.DIAGNOSTIC_MODULE
    (@INITVAL=  FALSE)
    (@SOURCES=
        (RunTimeValue   (FALSE))
    )
    (@CACTIONS=
        (LoadKB ("MSWITCH.tkb")  (@LEVEL=ENABLE;))
    )
)

(@SLOT= MSWITCH_CH21.NAME
    (@INITVAL=  "MSWITCH")
    (@SOURCES=
        (RunTimeValue   ("MSWITCH"))
    )
)

(@SLOT= MSWITCH_CH21.SENSOR_IN
    (@INITVAL=  "PM_2")
    (@SOURCES=
        (RunTimeValue   ("PM_2"))
    )
)

(@SLOT= MSWITCH_CH21.SENSOR_OUT
    (@INITVAL=  "PM_3")
    (@SOURCES=
        (RunTimeValue   ("PM_3"))
    )
)

(@SLOT= MSWITCH_CH22.DIAGNOSTIC_MODULE
    (@INITVAL=  FALSE)
    (@SOURCES=
        (RunTimeValue   (FALSE))
    )
    (@CACTIONS=
        (LoadKB ("MSWITCH.tkb")  (@LEVEL=ENABLE;))
    )
)

(@SLOT= MSWITCH_CH22.NAME
    (@INITVAL=  "MSWITCH")
    (@SOURCES=
        (RunTimeValue   ("MSWITCH"))
    )
)

(@SLOT= MSWITCH_CH22.SENSOR_IN
    (@INITVAL=  "PM_2")
    (@SOURCES=
        (RunTimeValue   ("PM_2"))
    )
)

(@SLOT= MSWITCH_CH22.SENSOR_OUT
    (@INITVAL=  "PM_4")
    (@SOURCES=
        (RunTimeValue   ("PM_4"))
    )
)

(@SLOT= PM_0.LEVEL
    (@INITVAL=  "OK")
    (@SOURCES=
        (RunTimeValue   ("OK"))
    )
)
```

```
(@SLOT= PM_0.NAME
    (@INITVAL=  "PM_0")
    (@SOURCES=
        (RunTimeValue   ("PM_0"))
    )
)

(@SLOT= PM_0.READING
    (@INITVAL=  "GOOD")
    (@SOURCES=
        (RunTimeValue   ("GOOD"))
    )
)

(@SLOT= PM_1.NAME
    (@INITVAL=  "PM_1")
    (@SOURCES=
        (RunTimeValue   ("PM_1"))
    )
)

(@SLOT= PM_2.NAME
    (@INITVAL=  "PM_2")
    (@SOURCES=
        (RunTimeValue   ("PM_2"))
    )
)

(@SLOT= PM_3.NAME
    (@INITVAL=  "PM_3")
    (@SOURCES=
        (RunTimeValue   ("PM_3"))
    )
)

(@SLOT= PM_4.NAME
    (@INITVAL=  "PM_4")
    (@SOURCES=
        (RunTimeValue   ("PM_4"))
    )
)

(@SLOT= PM_5.NAME
    (@INITVAL=  "PM_5")
    (@SOURCES=
        (RunTimeValue   ("PM_5"))
    )
)

(@SLOT= PM_6.NAME
    (@INITVAL=  "PM_6")
    (@SOURCES=
        (RunTimeValue   ("PM_6"))
    )
)

(@SLOT= PM_7.NAME
    (@INITVAL=  "PM_7")
    (@SOURCES=
        (RunTimeValue   ("PM_7"))
    )
)

(@SLOT= PM_8.NAME
    (@INITVAL=  "PM_8")
    (@SOURCES=
        (RunTimeValue   ("PM_8"))
    )
)
```

```
(@RULE= RULE_029__QUALIFICATION_OF_CONFIDENCE__REJECTED
    (@LHS=
        (<= (\CURRENT_FAULT.NAME\.CF)    (-0.9))
    )
    (@HYPO= Evaluate_Certainty_Factors)
    (@RHS=
        (Let     (\CURRENT_FAULT.NAME\.CONFIDENCE)     ("REJECTED"))
        (Let     (\CURRENT_FAULT.NAME\.VERIFIED)  (FALSE))
    )
)

(@RULE= RULE_028__QUALIFICATION_OF_CONFIDENCE__VERY_IMPROBABLE
    (@LHS=
        (<= (\CURRENT_FAULT.NAME\.CF)    (-0.75))
        (>  (\CURRENT_FAULT.NAME\.CF)    (-0.9))
    )
    (@HYPO= Evaluate_Certainty_Factors)
    (@RHS=
        (Let     (\CURRENT_FAULT.NAME\.CONFIDENCE)     ("VERY_IMPROBABLE"))
    )
)

(@RULE= RULE_027__QUALIFICATION_OF_CONFIDENCE__IMPROBABLE
    (@LHS=
        (<= (\CURRENT_FAULT.NAME\.CF)    (-0.5))
        (>  (\CURRENT_FAULT.NAME\.CF)    (-0.75))
    )
    (@HYPO= Evaluate_Certainty_Factors)
    (@RHS=
        (Let     (\CURRENT_FAULT.NAME\.CONFIDENCE)     ("IMPROBABLE"))
    )
)

(@RULE= RULE_026__QUALIFICATION_OF_CONFIDENCE__UNLIKELY
    (@LHS=
        (<= (\CURRENT_FAULT.NAME\.CF)    (-0.25))
        (>  (\CURRENT_FAULT.NAME\.CF)    (-0.5))
    )
    (@HYPO= Evaluate_Certainty_Factors)
    (@RHS=
        (Let     (\CURRENT_FAULT.NAME\.CONFIDENCE)     ("UNLIKELY"))
    )
)

(@RULE= RULE_025__QUALIFICATION_OF_CONFIDENCE__UNKNOWN
    (@LHS=
        (>  (\CURRENT_FAULT.NAME\.CF)    (-0.25))
        (<  (\CURRENT_FAULT.NAME\.CF)    (0.25))
    )
    (@HYPO= Evaluate_Certainty_Factors)
    (@RHS=
        (Let     (\CURRENT_FAULT.NAME\.CONFIDENCE)     ("UNKNOWN"))
    )
)

(@RULE= RULE_024__QUALIFICATION_OF_CONFIDENCE__POSSIBLE
    (@LHS=
        (>= (\CURRENT_FAULT.NAME\.CF)    (0.25))
        (<  (\CURRENT_FAULT.NAME\.CF)    (0.5))
    )
    (@HYPO= Evaluate_Certainty_Factors)
    (@RHS=
        (Let     (\CURRENT_FAULT.NAME\.CONFIDENCE)     ("POSSIBLE"))
    )
)
```

```
(@RULE= RULE_023__QUALIFICATION_OF_CONFIDENCE__LIKELY
    (@LHS=
        (>= (\CURRENT_FAULT.NAME\.CF)    (0.5))
        (<  (\CURRENT_FAULT.NAME\.CF)    (0.75))
    )
    (@HYPO= Evaluate_Certainty_Factors)
    (@RHS=
        (Let     (\CURRENT_FAULT.NAME\.CONFIDENCE)     ("LIKELY"))
    )
)

(@RULE= RULE_022__QUALIFICATION_OF_CONFIDENCE__PROBABLE
    (@LHS=
        (>= (\CURRENT_FAULT.NAME\.CF)    (0.75))
        (<  (\CURRENT_FAULT.NAME\.CF)    (0.9))
    )
    (@HYPO= Evaluate_Certainty_Factors)
    (@RHS=
        (Let     (\CURRENT_FAULT.NAME\.CONFIDENCE)     ("PROBABLE"))
    )
)

(@RULE= RULE_021__QUALIFICATION_OF_CONFIDENCE__ESTABLISHED
    (@LHS=
        (>= (\CURRENT_FAULT.NAME\.CF)    (0.9))
    )
    (@HYPO= Evaluate_Certainty_Factors)
    (@RHS=
        (Let     (\CURRENT_FAULT.NAME\.CONFIDENCE)     ("ESTABLISHED"))
        (Let     (\CURRENT_FAULT.NAME\.VERIFIED)  (TRUE))
    )
)

(@RULE= RULE_012__MODEL_MATRIX_SWITCH
    (@LHS=
        (Is (MSWITCH.CONFIG)     ("B"))
    )
    (@HYPO= Model_Matrix_Switch_SubSystem)
    (@RHS=
        (CreateObject    (MSWITCH_CH12)   (|SUBSYSTEMS|))
        (CreateObject    (MSWITCH_CH21)   (|SUBSYSTEMS|))
    )
)

(@RULE= RULE_011__MODEL_MATRIX_SWITCH
    (@LHS=
        (Is (MSWITCH.CONFIG)     ("A"))
    )
    (@HYPO= Model_Matrix_Switch_SubSystem)
    (@RHS=
        (CreateObject    (MSWITCH_CH11)   (|SUBSYSTEMS|))
        (CreateObject    (MSWITCH_CH22)   (|SUBSYSTEMS|))
    )
)

(@RULE= RULE_902__RETURN_LIST_OF_BAD_SENSORS_TO_ToolBook
    (@LHS=
        (Yes    (TBK_Request.Bad_Sensors))
    )
    (@HYPO= Return_BAD_Sensors)
    (@RHS=
        (Let    ({|BAD_SENSORS|}.RTN_READING)    (TRUE))
        (Strategy    (@CACTIONSON=FALSE;))
        (Reset   ({|BAD_SENSORS|}.RTN_READING))
        (Strategy    (@CACTIONSON=TRUE;))
        (Execute     ("BadSensorReadingsReturned"))
    )
)
```

```
(@RULE= RULE_901__RETRIEVE_SENSOR_PARAMETERS_FROM_SENSOR_DATABASE_AND_RETURN\
_NOMINAL_DATA_TO_ToolBook
    (@LHS=
        (Yes    (TBK_Request.Nominal_Sensor_Data))
        (Retrieve   ("SENSOR.nxp")   (@TYPE=NXPDB;@FWRD=FALSE;@UNKNOWN=TRUE;@PROPS=NAME,\
NOMINAL,TOLERANCE,ZERO_LEVEL;@FIELDS="NAME",\
"NOMINAL","TOLERANCE","ZERO_LEVEL";@ATOMS=<|SENSORS|>;\
))
    )
    (@HYPO= Return_Nominal_Sensor_Data)
    (@RHS=
        (Do (<|SENSORS|>.NOMINAL)    (<|SENSORS|>.NOMINAL))
    )
)

(@RULE= RULE_003__QUALIFICATION_OF_HIGH_SENSOR_LEVELS
    (@LHS=
        (>  (\CURRENT_SENSOR.NAME\.ERROR)    (0))
    )
    (@HYPO= Sensor_Level_Description.HIGH)
    (@RHS=
        (Let    (\CURRENT_SENSOR.NAME\.LEVEL)    ("HIGH"))
    )
)

(@RULE= RULE_004__QUALIFICATION_OF_LOW_SENSOR_LEVELS
    (@LHS=
        (<  (\CURRENT_SENSOR.NAME\.ERROR)    (0))
    )
    (@HYPO= Sensor_Level_Description.LOW)
    (@RHS=
        (Let    (\CURRENT_SENSOR.NAME\.LEVEL)    ("LOW"))
    )
)

(@RULE= RULE_005__QUALIFICATION_OF_OK_SENSOR_LEVELS
    (@LHS=
        (<= (ABS(\CURRENT_SENSOR.NAME\.ERROR)-\CURRENT_SENSOR.NAME\.TOLERANCE)    (0))
    )
    (@HYPO= Sensor_Level_Description.OK)
    (@RHS=
        (Let    (\CURRENT_SENSOR.NAME\.LEVEL)    ("OK"))
    )
)

(@RULE= RULE_006__QUALIFICATION_OF_ZERO_SENSOR_LEVELS
    (@LHS=
        (<= (\CURRENT_SENSOR.NAME\.DATA-\CURRENT_SENSOR.NAME\.ZERO_LEVEL)  (0))
    )
    (@HYPO= Sensor_Level_Description.ZERO)
    (@RHS=
        (Let    (\CURRENT_SENSOR.NAME\.LEVEL)    ("ZERO"))
    )
)

(@RULE= RULE_001__QUALIFICATION_OF_BAD_SENSOR_READINGS
    (@LHS=
        (>  (ABS(\CURRENT_SENSOR.NAME\.ERROR)-\CURRENT_SENSOR.NAME\.TOLERANCE)    (0))
    )
    (@HYPO= Sensor_Reading_Description.BAD)
    (@RHS=
        (Let    (\CURRENT_SENSOR.NAME\.READING)  ("BAD"))
        (CreateObject    (\CURRENT_SENSOR.NAME\)  (|BAD_SENSORS|))
        (Let    (\CURRENT_SENSOR.NAME\.EVALUATED)    (TRUE))
    )
)
```

```
(@RULE= RULE_002__QUALIFICATION_OF_GOOD_SENSOR_READINGS
    (@LHS=
        (<= (ABS(\CURRENT_SENSOR.NAME\.ERROR)-\CURRENT_SENSOR.NAME\.TOLERANCE)     (0))
    )
    (@HYPO= Sensor_Reading_Description.GOOD)
    (@RHS=
        (Let     (\CURRENT_SENSOR.NAME\.READING)   ("GOOD"))
        (Let     (\CURRENT_SENSOR.NAME\.EVALUATED)     (TRUE))
    )
)


(@GLOBALS=
    @INHVALUP=FALSE;
    @INHVALDOWN=TRUE;
    @INHOBJUP=FALSE;
    @INHOBJDOWN=FALSE;
    @INHCLASSUP=FALSE;
    @INHCLASSDOWN=TRUE;
    @INHBREADTH=TRUE;
    @INHPARENT=FALSE;
    @PWTRUE=TRUE;
    @PWFALSE=TRUE;
    @PWNOTKNOWN=TRUE;
    @EXHBWRD=TRUE;
    @PTGATES=TRUE;
    @PFACTIONS=TRUE;
    @SOURCESON=TRUE;
    @CACTIONSON=TRUE;
    @VOLLIST=PM_1.DATA;
)
```

APPENDIX B
FAULT DETECTION KNOWLEDGE BASE

```
(@VERSION=   020)

(@OBJECT=    A_Fault_Has_Been_Detected
    (@PROPERTIES=
        Value    @TYPE=Boolean;
    )
)

(@OBJECT=    A_Fault_Has_Not_Been_Detected
    (@PROPERTIES=
        Value    @TYPE=Boolean;
    )
)

(@OBJECT=    Transponder_Functioning_Properly
    (@PROPERTIES=
        Value    @TYPE=Boolean;
    )
)

(@RULE= R1
    (@LHS=
        (Yes     (TBK_Request.Detection))
        (Is (<¦SENSORS¦>.READING)    ("BAD"))
    )
    (@HYPO= A_Fault_Has_Been_Detected)
    (@RHS=
        (Execute     ("FaultDetected"))
    )
)

(@RULE= R2
    (@LHS=
        (Yes     (TBK_Request.Detection))
        (Is ({¦SENSORS¦}.READING)    ("GOOD"))
    )
    (@HYPO= A_Fault_Has_Not_Been_Detected)
    (@RHS=
        (Execute     ("NoFaultDetected"))
    )
)
```

APPENDIX C
FAULT ISOLATION KNOWLEDGE BASE

```
(@VERSION=  020)

(@OBJECT=   Isolate_Fault_Symptoms
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@RULE= RULE04__ISOLATION_OF_FAULT_TO_FREQUENCY_CONPONENTS
    (@LHS=
        (Yes    (TBK_Request.Isolation))
        (NotMember  ({|BAD_SENSORS|})    (<|PWR_SENSORS|>))
        (Is (<|BER_SENSORS|>.READING)    ("BAD"))
    )
    (@HYPO= Isolate_Fault_Symptoms)
)

(@RULE= RULE01__ISOLATION_OF_FAULT_TO_SUBSYSTEMS
    (@LHS=
        (Yes    (TBK_Request.Isolation))
        (Yes    (Model_Matrix_Switch_SubSystems))
        (Is (<|SUB_SYSTEMS|>.READING_IN) ("GOOD"))
        (Is (<|SUB_SYSTEMS|>.READING_OUT)    ("BAD"))
    )
    (@HYPO= Isolate_Fault_Symptoms)
    (@RHS=
        (Let    (<|SUB_SYSTEMS|>.ISOLATED)   (TRUE))
        (CreateObject   (<|SUB_SYSTEMS|>)    (|ISOLATED_SUB_SYSTEMS|))
    )
)
```

APPENDIX D
RECEIVER SUBSYSTEMS DIAGNOSTIC KNOWLEDGE BASES

D.1 CHANNEL 1 RECEIVER SUBSYSTEM

```
(@VERSION=   020)
(@PROPERTY=  CF  @TYPE=Float;)
(@PROPERTY=  COMPONENT    @TYPE=String;)
(@PROPERTY=  COMPONENT_IN     @TYPE=String;)
(@PROPERTY=  COMPONENT_OUT    @TYPE=String;)
(@PROPERTY=  CONFIDENCE  @TYPE=String;)
(@PROPERTY=  COUPLING     @TYPE=Boolean;)
(@PROPERTY=  GAIN    @TYPE=Float;)
(@PROPERTY=  INF_CAT @TYPE=Float;)
(@PROPERTY=  MB  @TYPE=Float;)
(@PROPERTY=  MB_ACCUM     @TYPE=Float;)
(@PROPERTY=  MD  @TYPE=Float;)
(@PROPERTY=  MD_ACCUM     @TYPE=Float;)
(@PROPERTY=  MODEL_GAIN   @TYPE=Float;)
(@PROPERTY=  MODEL_POWER_IN   @TYPE=Float;)
(@PROPERTY=  MODEL_POWER_OUT  @TYPE=Float;)
(@PROPERTY=  NAME    @TYPE=String;)
(@PROPERTY=  NOMINAL_GAIN     @TYPE=Float;)
(@PROPERTY=  NOMINAL_POWER_OUT    @TYPE=Float;)
(@PROPERTY=  NOMINAL_SETTING  @TYPE=Float;)
(@PROPERTY=  NONINAL_POWER_IN     @TYPE=Float;)
(@PROPERTY=  POWER_IN     @TYPE=Float;)
(@PROPERTY=  POWER_LEVEL_IN   @TYPE=String;)
(@PROPERTY=  POWER_LEVEL_OUT  @TYPE=String;)
(@PROPERTY=  POWER_OUT    @TYPE=Float;)
(@PROPERTY=  SETTING @TYPE=Float;)
(@PROPERTY=  VERIFIED     @TYPE=Boolean;)


(@CLASS=     AMP_FAULTS
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@CLASS=     AMPLIFIERS
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        GAIN
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NOMINAL_GAIN
        NOMINAL_POWER_OUT
        NONINAL_POWER_IN
        POWER_IN
        POWER_LEVEL_IN
        POWER_LEVEL_OUT
        POWER_OUT
    )
)

(@CLASS=     ATTEN_FAULTS
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@CLASS=     ATTENUATORS
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        GAIN
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NOMINAL_GAIN
        NOMINAL_POWER_OUT
        NOMINAL_SETTING
        NONINAL_POWER_IN
        POWER_IN
        POWER_LEVEL_IN
        POWER_LEVEL_OUT
        POWER_OUT
        SETTING
    )
)

(@CLASS=    COMP_COUPLING_FAULT_STATES
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@CLASS=    COMPONENTS
    (@SUBCLASSES=
        ATTENUATORS
        AMPLIFIERS
        RECEIVERS
        LOCAL_OSCILATORS
    )
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        GAIN
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NOMINAL_GAIN
        NOMINAL_POWER_OUT
        NONINAL_POWER_IN
        POWER_IN
        POWER_OUT
    )
)

(@CLASS=    FAULT_STATES
    (@SUBCLASSES=
        ATTEN_FAULTS
        AMP_FAULTS
        RECEIVER_FAULTS
        LO_FAULTS
        COMP_COUPLING_FAULT_STATES
    )
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@CLASS=    LEVEL_1_FAULT_STATES
    (@PROPERTIES=
        VERIFIED
    )
)

(@CLASS=    LO_FAULTS
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@CLASS=    LOCAL_OSCILATORS
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        GAIN
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NOMINAL_GAIN
        NOMINAL_POWER_OUT
        NONINAL_POWER_IN
        POWER_IN
        POWER_LEVEL_IN
        POWER_LEVEL_OUT
        POWER_OUT
    )
)

(@CLASS=    RECEIVER_FAULTS
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@CLASS=    RECEIVERS
    (@PROPERTIES=
        COMPONENT_IN
        COMPONENT_OUT
        GAIN
        MODEL_GAIN
        MODEL_POWER_IN
        MODEL_POWER_OUT
        NAME
        NOMINAL_GAIN
        NOMINAL_POWER_OUT
        NONINAL_POWER_IN
        POWER_IN
        POWER_LEVEL_IN
        POWER_LEVEL_OUT
        POWER_OUT
    )
)

(@CLASS=    UNCERTAINTY_OVERHEAD
    (@SUBCLASSES=
        FAULT_STATES
    )
    (@PROPERTIES=
        CF
        CONFIDENCE
        MB
        MB_ACCUM
        MD
        MD_ACCUM
    )
)

(@OBJECT=   AMP_COUPLING
    (@CLASSES=
        AMP_FAULTS
        COMP_COUPLING_FAULT_STATES
    )
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@OBJECT=   AMP_GENERAL_FAILURE
    (@CLASSES=
        AMP_FAULTS
    )
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@OBJECT=   ATTEN_COUPLING
    (@CLASSES=
        ATTEN_FAULTS
        COMP_COUPLING_FAULT_STATES
    )
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@OBJECT=   ATTEN_GENERAL_FAILURE
    (@CLASSES=
        ATTEN_FAULTS
    )
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@OBJECT=   ATTEN_SETTING
    (@CLASSES=
        ATTEN_FAULTS
    )
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@OBJECT=   CURRENT_COMPONENT
    (@PROPERTIES=
        COUPLING
        NAME
    )
)

(@OBJECT=   CURRENT_FAULT
    (@PROPERTIES=
        NAME
    )
)

(@OBJECT=   CURRENT_SUBSYSTEM
    (@PROPERTIES=
        POWER_LEVEL_OUT
    )
)

(@OBJECT=   Develop_Diagnostic_Strategy
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@OBJECT=   Evaluate_Attenuator_Setting
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@OBJECT=   Evaluate_Fault_State_Confidence_Factors
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
```

```
)

(@OBJECT=    FN
    (@CLASSES=
        FAULT_STATES
    )
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@OBJECT=    FO
    (@CLASSES=
        FAULT_STATES
    )
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@OBJECT=    FP
    (@CLASSES=
        FAULT_STATES
    )
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@OBJECT=    FQ
    (@CLASSES=
        FAULT_STATES
    )
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@OBJECT=    FR
    (@CLASSES=
        FAULT_STATES
    )
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@OBJECT=    FS
    (@CLASSES=
        FAULT_STATES
    )
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@OBJECT=    FT
    (@CLASSES=
        FAULT_STATES
    )
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@OBJECT=    FU
    (@CLASSES=
        FAULT_STATES
    )
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@OBJECT=   FV
    (@CLASSES=
        FAULT_STATES
    )
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@OBJECT=   FW
    (@CLASSES=
        FAULT_STATES
    )
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@OBJECT=   FX
    (@CLASSES=
        FAULT_STATES
    )
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@OBJECT=   FY
    (@CLASSES=
        FAULT_STATES
    )
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@OBJECT=   FZ
    (@CLASSES=
        FAULT_STATES
    )
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@OBJECT=   Initialize_Database
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@OBJECT=   Level_1_Diagnostics
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@OBJECT=   LO_COUPLING
    (@CLASSES=
        COMP_COUPLING_FAULT_STATES
        LO_FAULTS
    )
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@OBJECT=   LO_GENERAL_FAILURE
    (@CLASSES=
        LO_FAULTS
    )
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@OBJECT=   OPEN_GATE
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@OBJECT=   RCVR_COUPLING
    (@CLASSES=
        COMP_COUPLING_FAULT_STATES
        RECEIVER_FAULTS
    )
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@OBJECT=   RCVR_GENERAL_FAILURE
    (@CLASSES=
        RECEIVER_FAULTS
    )
    (@PROPERTIES=
        CF
        COMPONENT
        CONFIDENCE
        INF_CAT
        MB
        MB_ACCUM
        MD
        MD_ACCUM
        NAME
        POWER_LEVEL_OUT
        VERIFIED
    )
)

(@OBJECT=   Test_Component_Coupling
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@SLOT= AMP_FAULTS.COMPONENT
    (@INITVAL=  "AMP_1")
    (@SOURCES=
        (RunTimeValue   ("AMP_1"))
    )
)

(@SLOT= ATTEN_FAULTS.COMPONENT
    (@INITVAL=  "SA12")
    (@SOURCES=
        (RunTimeValue   ("SA12"))
    )
)

(@SLOT= ATTEN_FAULTS.VERIFIED
    (@SOURCES=
        (Do (SELF.NAME) (CURRENT_FAULT.NAME))
        (Do (SELF.COMPONENT)
(CURRENT_COMPONENT.NAME))
        (Reset  (Evaluate_Attenuator_Setting))
        (Do (Evaluate_Attenuator_Setting)
(Evaluate_Attenuator_Setting))
    )
)

(@SLOT= COMP_COUPLING_FAULT_STATES.VERIFIED
    (@SOURCES=
        (Do (SELF.NAME) (CURRENT_FAULT.NAME))
        (Do (SELF.COMPONENT)
(CURRENT_COMPONENT.NAME))
        (Reset  (Test_Component_Coupling))
        (Do (Test_Component_Coupling)
(Test_Component_Coupling))
    )
)

(@SLOT= COMPONENTS.GAIN
    (@SOURCES=
        (Do (SELF.POWER_OUT-SELF.POWER_IN)
(SELF.GAIN))
    )
    (@CACTIONS=
        (Do (SELF.POWER_IN+SELF.GAIN)
(SELF.POWER_OUT))
    )
)

(@SLOT= COMPONENTS.MODEL_GAIN
    (@SOURCES=
        (Do (SELF.NOMINAL_GAIN) (SELF.MODEL_GAIN))
    )
    (@CACTIONS=
        (Do (SELF.MODEL_POWER_IN+SELF.MODEL_GAIN)
(SELF.MODEL_POWER_OUT))
    )
)

(@SLOT= COMPONENTS.MODEL_POWER_IN
    (@SOURCES=
        (Do (\SELF.COMPONENT_IN\.MODEL_POWER_OUT)
(SELF.MODEL_POWER_IN))
    )
    (@CACTIONS=
        (Do (SELF.MODEL_POWER_IN+SELF.MODEL_GAIN)
(SELF.MODEL_POWER_OUT))
    )
)

(@SLOT= COMPONENTS.MODEL_POWER_OUT
    (@SOURCES=
        (Do (SELF.MODEL_POWER_IN+SELF.MODEL_GAIN)
(SELF.MODEL_POWER_OUT))
    )
    (@CACTIONS=
        (Do (SELF.MODEL_POWER_OUT)
(\SELF.COMPONENT_OUT\.MODEL_POWER_IN))
    )
)

(@SLOT= COMPONENTS.POWER_IN
    (@SOURCES=
        (Do (\SELF.COMPONENT_IN\.POWER_OUT)
(SELF.POWER_IN))
    )
    (@CACTIONS=
        (Do (SELF.POWER_IN+SELF.GAIN)
(SELF.POWER_OUT))
```

```
    )
)

(@SLOT= COMPONENTS.POWER_OUT
    (@SOURCES=
        (Do (SELF.POWER_IN+SELF.GAIN)
(SELF.POWER_OUT))
    )
    (@CACTIONS=
        (Do (SELF.POWER_OUT)
(\SELF.COMPONENT_OUT\.POWER_IN))
    )
)

(@SLOT= FAULT_STATES.CF
    (@CACTIONS=
        (Do (SELF.NAME) (CURRENT_FAULT.NAME))
        (Reset
(Evaluate_Fault_State_Confidence_Factors))
        (Do
(Evaluate_Fault_State_Confidence_Factors)
(Evaluate_Fault_State_Confidence_Factors))
    )
)

(@SLOT= LO_FAULTS.COMPONENT
    (@INITVAL=   "RCVRLO")
    (@SOURCES=
        (RunTimeValue    ("RCVRLO"))
    )
)

(@SLOT= RECEIVER_FAULTS.COMPONENT
    (@INITVAL=   "RCVR_1")
    (@SOURCES=
        (RunTimeValue    ("RCVR_1"))
    )
)

(@SLOT= SENSORS.LEVEL
    (@SOURCES=
        (Do (SELF.NAME) (CURRENT_SENSOR.NAME))
        (Reset  (Sensor_Level_Description.HIGH))
        (Do (Sensor_Level_Description.HIGH)
(Sensor_Level_Description.HIGH))
        (Reset  (Sensor_Level_Description.ZERO))
        (Do (Sensor_Level_Description.ZERO)
(Sensor_Level_Description.ZERO))
        (Reset  (Sensor_Level_Description.LOW))
        (Do (Sensor_Level_Description.LOW)
(Sensor_Level_Description.LOW))
    )
)

(@SLOT= UNCERTAINTY_OVERHEAD.MB
    @COMMENTS="This the Measure of Belief (MB) slot
for  all  objects  utilizing  the  uncertainty
overhead";
    (@INITVAL=   0.0)
    (@SOURCES=
        (RunTimeValue    (0.0))
    )
    (@CACTIONS=
        (Do
(SELF.MB_ACCUM+(1-SELF.MB_ACCUM)*SELF.MB)
(SELF.MB_ACCUM))
        (Reset  (SELF.MB))
    )
)

(@SLOT= UNCERTAINTY_OVERHEAD.MB_ACCUM
    (@INITVAL=   0.0)
    (@SOURCES=
        (RunTimeValue    (0.0))
    )
    (@CACTIONS=
        (Do (SELF.MB_ACCUM-SELF.MD_ACCUM)
(SELF.CF))
    )
)

(@SLOT= UNCERTAINTY_OVERHEAD.MD
    (@INITVAL=   0.0)
    (@SOURCES=
        (RunTimeValue    (0.0))
    )
    (@CACTIONS=
        (Do
(SELF.MD_ACCUM+(1-SELF.MD_ACCUM)*SELF.MD)
(SELF.MD_ACCUM))
        (Reset  (SELF.MD))
    )
)

(@SLOT= UNCERTAINTY_OVERHEAD.MD_ACCUM
    (@INITVAL=   0.0)
    (@SOURCES=
        (RunTimeValue    (0.0))
    )
    (@CACTIONS=
        (Do (SELF.MB_ACCUM-SELF.MD_ACCUM)
(SELF.CF))
    )
)

(@SLOT= AMP_COUPLING.NAME
    (@INITVAL=   "AMP_COUPLING")
    (@SOURCES=
        (RunTimeValue    ("AMP_COUPLING"))
    )
)

(@SLOT= AMP_COUPLING.POWER_LEVEL_OUT
    (@INITVAL=   "ZERO")
    (@SOURCES=
        (RunTimeValue    ("ZERO"))
    )
)

(@SLOT= AMP_COUPLING.VERIFIED
    @INFATOM=AMP_COUPLING.INF_CAT;
)

(@SLOT= AMP_GENERAL_FAILURE.NAME
    (@INITVAL=   "AMP_GENERAL_FAILURE")
    (@SOURCES=
        (RunTimeValue    ("AMP_GENERAL_FAILURE"))
    )
)

(@SLOT= AMP_GENERAL_FAILURE.POWER_LEVEL_OUT
    (@INITVAL=   "HIGH, LOW, ZERO")
    (@SOURCES=
        (RunTimeValue    ("HIGH, LOW, ZERO"))
    )
)

(@SLOT= AMP_GENERAL_FAILURE.VERIFIED
    @INFATOM=AMP_GENERAL_FAILURE.INF_CAT;
)

(@SLOT= ATTEN_COUPLING.NAME
    (@INITVAL=   "ATTEN_COUPLING")
    (@SOURCES=
        (RunTimeValue    ("ATTEN_COUPLING"))
    )
```

```
)

(@SLOT= ATTEN_COUPLING.POWER_LEVEL_OUT
    (@INITVAL=  "ZERO")
    (@SOURCES=
        (RunTimeValue   ("ZERO"))
    )
)

(@SLOT= ATTEN_COUPLING.VERIFIED
    @INFATOM=ATTEN_COUPLING.INF_CAT;
)

(@SLOT= ATTEN_GENERAL_FAILURE.NAME
    (@INITVAL=  "ATTEN_GENERAL_FAILURE")
    (@SOURCES=
        (RunTimeValue   ("ATTEN_GENERAL_FAILURE"))
    )
)

(@SLOT= ATTEN_GENERAL_FAILURE.POWER_LEVEL_OUT
    (@INITVAL=  "HIGH, LOW, ZERO")
    (@SOURCES=
        (RunTimeValue   ("HIGH, LOW, ZERO"))
    )
)

(@SLOT= ATTEN_GENERAL_FAILURE.VERIFIED
    @INFATOM=ATTEN_GENERAL_FAILURE.INF_CAT;
)

(@SLOT= ATTEN_SETTING.NAME
    (@INITVAL=  "ATTEN_SETTING")
    (@SOURCES=
        (RunTimeValue   ("ATTEN_SETTING"))
    )
)

(@SLOT= ATTEN_SETTING.POWER_LEVEL_OUT
    (@INITVAL=  "HIGH, LOW, ZERO")
    (@SOURCES=
        (RunTimeValue   ("HIGH, LOW, ZERO"))
    )
)

(@SLOT= ATTEN_SETTING.VERIFIED
    @INFATOM=ATTEN_SETTING.INF_CAT;
)

(@SLOT= CURRENT_COMPONENT.COUPLING
    @PROMPT="Check   the   coupling   of
@V(CURRENT_COMPONENT.NAME). Is the input or output
connection loose ?";
    @COMMENTS="This  slot  will  implemented  by
ToolBook";@WHY="It   is   possible   that
@V(CURRENT_COMPONENT)  is  not  coupled  to  the
transponder correctly.";
    (@SOURCES=
        (AskQuestion
(CURRENT_COMPONENT.COUPLING) (NOTKNOWN))
    )
)

(@SLOT= CURRENT_SUBSYSTEM.POWER_LEVEL_OUT
    (@INITVAL=  "ZERO")
    (@SOURCES=
        (RunTimeValue   ("ZERO"))
    )
)

(@SLOT= FN.VERIFIED
    @INFATOM=FN.INF_CAT;
)

(@SLOT= FO.VERIFIED
    @INFATOM=FO.INF_CAT;
)

(@SLOT= FP.VERIFIED
    @INFATOM=FP.INF_CAT;
)

(@SLOT= FQ.VERIFIED
    @INFATOM=FQ.INF_CAT;
)

(@SLOT= FR.VERIFIED
    @INFATOM=FR.INF_CAT;
)

(@SLOT= FS.VERIFIED
    @INFATOM=FS.INF_CAT;
)

(@SLOT= FT.VERIFIED
    @INFATOM=FT.INF_CAT;
)

(@SLOT= FU.VERIFIED
    @INFATOM=FU.INF_CAT;
)

(@SLOT= FV.VERIFIED
    @INFATOM=FV.INF_CAT;
)

(@SLOT= FW.VERIFIED
    @INFATOM=FW.INF_CAT;
)

(@SLOT= FX.VERIFIED
    @INFATOM=FX.INF_CAT;
)

(@SLOT= FY.VERIFIED
    @INFATOM=FY.INF_CAT;
)

(@SLOT= FZ.VERIFIED
    @INFATOM=FZ.INF_CAT;
)

(@SLOT= LO_COUPLING.VERIFIED
    @INFATOM=LO_COUPLING.INF_CAT;
)

(@SLOT= LO_GENERAL_FAILURE.VERIFIED
    @INFATOM=LO_GENERAL_FAILURE.INF_CAT;
)

(@SLOT= OPEN_GATE
    @PROMPT="Something Is Wrong.  The OPEN_GATE is
closed.  Please enter TRUE to continue ...";
    @COMMENTS="The  OPEN_GATE  is  a  boolean  slot
which is always true. ";@WHY="The OPEN_GATE should
always be TRUE.  It is used as a condition of rules
which must always fire to effect LHS actions.";
    (@INITVAL=  TRUE)
    (@SOURCES=
        (RunTimeValue   (TRUE))
    )
)

(@SLOT= PM_1.NAME
    (@INITVAL=  "PM_1")
    (@SOURCES=
        (RunTimeValue   ("PM_1"))
    )
)

(@SLOT= PM_1.ZERO_LEVEL
    (@INITVAL=  -30.0)
    (@SOURCES=
        (RunTimeValue   (-30.0))
    )
)

(@SLOT= RCVR_COUPLING.VERIFIED
    @INFATOM=RCVR_COUPLING.INF_CAT;
)

(@SLOT= RCVR_GENERAL_FAILURE.VERIFIED
    @INFATOM=RCVR_GENERAL_FAILURE.INF_CAT;
)

(@RULE= RULE01__DEVELOP_DIAGNOSTIC_STRATEGY
    @COMMENTS="This rule tests the signal power
level symptoms of all fault states against the
obserrved level.  All matches are created in class
of Level 1 Fault States.";@WHY="A diagnostic
strategy must be developed before Level 1
Diagnostics can begi
n.";
    (@LHS=
        (Retrieve   ("CH1RCV.nxp")
(@TYPE=NXPDB;@FILL=ADD;@FWRD=FALSE;@UNKNOWN=TRUE;\
@NAME="!Name!";@PROPS=INF_CAT;@FIELDS="INF_CAT";\
))
        (Execute    ("TestMultiValue")
(@ATOMID=<|FAULT_STATES|>.POWER_LEVEL_OUT;\
@STRING="@SUPERSET,@TEST=@V(CURRENT_SUBSYSTEM.POWE
R_LEVEL_OUT),\
@RETURN=LEVEL_1_FAULT_STATES,@COMP=STRING";\
))
    )
    (@HYPO= Develop_Diagnostic_Strategy)
    (@RHS=
        (Do (Level_1_Diagnostics)
(Level_1_Diagnostics))
    )
)

(@RULE=
RULE11__EVALUATE_REJECTED_CONFIDENCE_FACTORS
    (@LHS=
        (<= (\CURRENT_FAULT.NAME\.CF)    (-0.9))
    )
    (@HYPO=
Evaluate_Fault_State_Confidence_Factors)
    (@RHS=
        (Let    (\CURRENT_FAULT.NAME\.CONFIDENCE)
("REJECTED"))
        (Let    (\CURRENT_FAULT.NAME\.VERIFIED)
(FALSE))
    )
)

(@RULE=
RULE10__EVALUATE_IMPROBABLE_CONFIDENCE_FACTORS
    (@LHS=
        (<= (\CURRENT_FAULT.NAME\.CF)    (-0.75))
        (>  (\CURRENT_FAULT.NAME\.CF)    (-0.9))
    )
    (@HYPO=
Evaluate_Fault_State_Confidence_Factors)
    (@RHS=
        (Let    (\CURRENT_FAULT.NAME\.CONFIDENCE)
("IMPROBABLE"))
    )
)
```

```
(@RULE=
RULE09__EVALUATE_UNLIKELY_CONFIDENCE_FACTORS
    (@LHS=
        (<= (\CURRENT_FAULT.NAME\.CF)    (-0.5))
        (>  (\CURRENT_FAULT.NAME\.CF)    (-0.75))
    )
    (@HYPO=
Evaluate_Fault_State_Confidence_Factors)
    (@RHS=
        (Let    (\CURRENT_FAULT.NAME\.CONFIDENCE)
("UNLIKELY"))
    )
)

(@RULE= RULE08__EVALUATE_CONFIDENCE_FACTORS
    (@LHS=
        (<= (\CURRENT_FAULT.NAME\.CF)    (-0.25))
        (>  (\CURRENT_FAULT.NAME\.CF)    (-0.5))
    )
    (@HYPO=
Evaluate_Fault_State_Confidence_Factors)
    (@RHS=
        (Let    (\CURRENT_FAULT.NAME\.CONFIDENCE)(
"))
    )
)

(@RULE= RULE07__EVALUATE_UNKNOWN_CONFIDENCE_FACTORS
    (@LHS=
        (>  (\CURRENT_FAULT.NAME\.CF)    (-0.25))
        (<  (\CURRENT_FAULT.NAME\.CF)    (0.25))
    )
    (@HYPO=
Evaluate_Fault_State_Confidence_Factors)
    (@RHS=
        (Let    (\CURRENT_FAULT.NAME\.CONFIDENCE)
("UNKNOWN"))
    )
)

(@RULE= RULE06__EVALUATE_CONFIDENCE_FACTORS
    (@LHS=
        (>= (\CURRENT_FAULT.NAME\.CF)    (0.25))
        (<  (\CURRENT_FAULT.NAME\.CF)    (0.5))
    )
    (@HYPO=
Evaluate_Fault_State_Confidence_Factors)
    (@RHS=
        (Let    (\CURRENT_FAULT.NAME\.CONFIDENCE)(
"))
    )
)

(@RULE=
RULE05__EVALUATE_POSSIBLE_CONFIDENCE_FACTORS
    (@LHS=
        (>= (\CURRENT_FAULT.NAME\.CF)    (0.5))
        (<  (\CURRENT_FAULT.NAME\.CF)    (0.75))
    )
    (@HYPO=
Evaluate_Fault_State_Confidence_Factors)
    (@RHS=
        (Let    (\CURRENT_FAULT.NAME\.CONFIDENCE)
("POSSIBLE"))
    )
)

(@RULE=
RULE04__EVALUATE_PROBABLE_CONFIDENCE_FACTORS
    (@LHS=
        (>= (\CURRENT_FAULT.NAME\.CF)    (0.75))
        (<  (\CURRENT_FAULT.NAME\.CF)    (0.9))
    )
    (@HYPO=
Evaluate_Fault_State_Confidence_Factors)
    (@RHS=
        (Let    (\CURRENT_FAULT.NAME\.CONFIDENCE)
("PROBABLE"))
    )
)

(@RULE=
RULE03__EVALUATE_ESTABLISHED_CONFIDENCE_FACTORS
    (@LHS=
        (>= (\CURRENT_FAULT.NAME\.CF)    (0.9))
    )
    (@HYPO=
Evaluate_Fault_State_Confidence_Factors)
    (@RHS=
        (Let    (\CURRENT_FAULT.NAME\.CONFIDENCE)
("ESTABLISHED"))
        (Let    (\CURRENT_FAULT.NAME\.VERIFIED)
(TRUE))
    )
)

(@RULE= RULE99__INITIALIZE_LEARNING_DATABASE
    @COMMENTS="This rule initializes the infefence
categorys of all fault states to 1.0 in the
CH1RCV.nxp database";@WHY="At times in may be
necessary to forget everything learned about the
diagnostic strategy.";
    (@LHS=
        (Yes    (OPEN_GATE))
    )
    (@HYPO= Initialize_Database)
    (@RHS=
        (Do (1.0)    (<|FAULT_STATES|>.INF_CAT))
        (Write  ("CH1RCV.nxp")
(@TYPE=NXPDB;@FILL=NEW;@UNKNOWN=TRUE;@NAME="<|FAUL
T_STATES|>";\
@PROPS=INF_CAT;@FIELDS="INF_CAT";))
    )
)

(@RULE= RULE02__PURSUE_LEVEL_1_DIAGNOSTIC_STRATEGY
    @COMMENTS="This rule effects Level 1
Diagnostics by placing all fault states on the
agenda";@WHY="Level 1 Diagnostics is the first step
in the diagnostic process";
    (@LHS=
        (Yes
(<|LEVEL_1_FAULT_STATES|>.VERIFIED))
    )
    (@HYPO= Level_1_Diagnostics)
)

(@RULE= RULE03__QUALIFICATION_OF_SENSOR_LEVEL
    (@LHS=
        (>= (\CURRENT_SENSOR.NAME\.ERROR)    (0))
    )
    (@HYPO= Sensor_Level_Description.HIGH)
    (@RHS=
        (Let    (\CURRENT_SENSOR.NAME\.LEVEL)
("HIGH"))
    )
)

(@RULE= RULE20__GENERIC_TEST_FOR_COMPONENT_COUPLING
    @COMMENTS="This rule tests the signal power
level symptoms of all fault states against the
obserrved level.  All matches are created in class
of Level 1 Fault States.";@WHY="A diagnostic
strategy must be developed before Level 1
Diagnostics can begi
n.";
    (@LHS=
        (Reset  (CURRENT_COMPONENT.COUPLING))
        (Yes    (CURRENT_COMPONENT.COUPLING))
    )
    (@HYPO= Test_Component_Coupling)
    (@RHS=
        (Do (0.9)    (\CURRENT_FAULT.NAME\.MB))
    )
)
```

D.2 CHANNEL 2 RECEIVER SUBSYSTEM

```
(@VERSION=   020)
(@PROPERTY=  DESCRIPTION  @TYPE=String;)
(@PROPERTY=  INF_CATEGORY     @TYPE=Float;)
(@PROPERTY=  POWER_LEVEL_OUT  @TYPE=String;)
(@PROPERTY=  VERIFIED      @TYPE=Boolean;)


(@CLASS=     FAULT_STATES
    (@SUBCLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@CLASS=     HIGH_POWER_FAULT_STATES
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@CLASS=     LOW_POWER_FAULT_STATES
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@CLASS=     SUBSYSTEMS
    (@PROPERTIES=
        POWER_LEVEL_OUT
    )
)

(@CLASS=     ZERO_POWER_FAULT_STATES
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)


(@OBJECT=    Diagnose_HIGH_Power_Fault_States
    (@PROPERTIES=
        Value    @TYPE=Boolean;
    )
)

(@OBJECT=    Diagnose_LOW_Power_Fault_States
    (@PROPERTIES=
        Value    @TYPE=Boolean;
    )
)

(@OBJECT=    Diagnose_ZERO_Power_Fault_States
    (@PROPERTIES=
        Value    @TYPE=Boolean;
    )
)

(@OBJECT=    IFPC_Amplifier_Coupling_to_System
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=    IFPC_Amplifier_General_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=    IFPC_Amplifier_Power_Supply_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=    IFPC_Attenuator_Coupling_to_System
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=    IFPC_Attenuator_General_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)
```

```
(@OBJECT=   IFPC_Attenuator_Setting_Is_Incorrect
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
Local_Oscillator_Frequency_Setting_Is_Incorrect
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Local_Oscillator_General_Failure
    (@CLASSES=
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
Local_Oscillator_Out_of_Phase_Lock__Alarm
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
Local_Oscillator_Out_of_Phase_Lock__Test
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Local_Oscillator_Power_Supply_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
Local_Oscillator_Signal_Power_Level_Is_LOW
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Receiver_Unit_Coupling_to_System
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Receiver_Unit_General_Failure
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
Receiver_Unit_Primary_Power_Supply_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
Receiver_Unit_Secondary_Power_Supply_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Undefined_General_Failure
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
```

```
         VERIFIED
     )
)

(@OBJECT=
ZZ_Outstanding_Fault_States_for_Channel_2_Receiver
_System
     (@CLASSES=
         HIGH_POWER_FAULT_STATES
         LOW_POWER_FAULT_STATES
         ZERO_POWER_FAULT_STATES
     )
     (@PROPERTIES=
         DESCRIPTION
         INF_CATEGORY
         VERIFIED
     )
)

(@SLOT= IFPC_Amplifier_Coupling_to_System.VERIFIED

@INFATOM=IFPC_Amplifier_Coupling_to_System.INF_CAT
EGORY;
)

(@SLOT= IFPC_Amplifier_General_Failure.VERIFIED

@INFATOM=IFPC_Amplifier_General_Failure.INF_CATEGO
RY;
)

(@SLOT=
IFPC_Amplifier_Power_Supply_Failure.VERIFIED

@INFATOM=IFPC_Amplifier_Power_Supply_Failure.INF_C
ATEGORY;
)

(@SLOT= IFPC_Attenuator_Coupling_to_System.VERIFIED

@INFATOM=IFPC_Attenuator_Coupling_to_System.INF_CA
TEGORY;
)

(@SLOT= IFPC_Attenuator_General_Failure.VERIFIED

@INFATOM=IFPC_Attenuator_General_Failure.INF_CATEG
ORY;
)

(@SLOT=
IFPC_Attenuator_Setting_Is_Incorrect.VERIFIED

@INFATOM=IFPC_Attenuator_Setting_Is_Incorrect.INF_
CATEGORY;
)

(@SLOT=
Local_Oscillator_Frequency_Setting_Is_Incorrect.VE
RIFIED

@INFATOM=Local_Oscillator_Frequency_Setting_Is_Inc
orrect.INF_CATEGORY;
)

(@SLOT= Local_Oscillator_General_Failure.VERIFIED

@INFATOM=Local_Oscillator_General_Failure.INF_CATE
GORY;
)

(@SLOT=
Local_Oscillator_Out_of_Phase_Lock__Alarm.VERIFIED

@INFATOM=Local_Oscillator_Out_of_Phase_Lock__Alarm
.INF_CATEGORY;
)

(@SLOT=
Local_Oscillator_Out_of_Phase_Lock__Test.VERIFIED

@INFATOM=Local_Oscillator_Out_of_Phase_Lock__Test.
INF_CATEGORY;
)

(@SLOT=
Local_Oscillator_Power_Supply_Failure.VERIFIED

@INFATOM=Local_Oscillator_Power_Supply_Failure.INF
_CATEGORY;
)

(@SLOT=
Local_Oscillator_Signal_Power_Level_Is_LOW.VERIFIED

@INFATOM=Local_Oscillator_Signal_Power_Level_Is_LO
W.INF_CATEGORY;
)

(@SLOT= Receiver_Unit_Coupling_to_System.VERIFIED

@INFATOM=Receiver_Unit_Coupling_to_System.INF_CATE
GORY;
)

(@SLOT= Receiver_Unit_General_Failure.VERIFIED

@INFATOM=Receiver_Unit_General_Failure.INF_CATEGOR
Y;
)

(@SLOT=
Receiver_Unit_Primary_Power_Supply_Failure.VERIFIED

@INFATOM=Receiver_Unit_Primary_Power_Supply_Failur
e.INF_CATEGORY;
)

(@SLOT=
Receiver_Unit_Secondary_Power_Supply_Failure.VERIF
IED

@INFATOM=Receiver_Unit_Secondary_Power_Supply_Fail
ure.INF_CATEGORY;
)

(@SLOT= Undefined_General_Failure.VERIFIED

@INFATOM=Undefined_General_Failure.INF_CATEGORY;
)

(@SLOT=
ZZ_Outstanding_Fault_States_for_Channel_2_Receiver
_System.VERIFIED

@INFATOM=ZZ_Outstanding_Fault_States_for_Channel_2
_Receiver_System.INF_CATEGO\
RY;
)

(@RULE= R1
     (@LHS=
         (Is (<|SUBSYSTEMS|>.POWER_LEVEL_OUT) (HIGH))
         (Yes
(<|HIGH_POWER_FAULT_STATES|>.VERIFIED))
     )
```

```
    (@HYPO= Diagnose_HIGH_Power_Fault_States)
)

(@RULE= R2
    (@LHS=
        (Is (<¦SUBSYSTEMS¦>.POWER_LEVEL_OUT)
("LOW"))
        (Yes
(<¦LOW_POWER_FAULT_STATES¦>.VERIFIED))
    )
    (@HYPO= Diagnose_LOW_Power_Fault_States)
)

(@RULE= R3
    (@LHS=
        (Is (<¦SUBSYSTEMS¦>.POWER_LEVEL_OUT)
("ZERO"))
        (Yes
(<¦ZERO_POWER_FAULT_STATES¦>.VERIFIED))
    )
    (@HYPO= Diagnose_ZERO_Power_Fault_States)
)
```

APPENDIX E
MATRIX SWITCH SUBSYSTEM DIAGNOSTIC KNOWLEDGE BASE

```
(@VERSION=   020)
(@PROPERTY=  DESCRIPTION  @TYPE=String;)
(@PROPERTY=  INF_CATEGORY     @TYPE=Float;)
(@PROPERTY=  POWER_LEVEL_OUT  @TYPE=String;)
(@PROPERTY=  VERIFIED     @TYPE=Boolean;)


(@CLASS=     FAULT_STATES
    (@SUBCLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@CLASS=     HIGH_POWER_FAULT_STATES
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@CLASS=     LOW_POWER_FAULT_STATES
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@CLASS=     SUBSYSTEMS
    (@PROPERTIES=
        POWER_LEVEL_OUT
    )
)

(@CLASS=     ZERO_POWER_FAULT_STATES
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=    Diagnose_HIGH_Power_Fault_States
    (@PROPERTIES=
        Value    @TYPE=Boolean;
    )
)

(@OBJECT=    Diagnose_LOW_Power_Fault_States
    (@PROPERTIES=
        Value    @TYPE=Boolean;
    )
)

(@OBJECT=    Diagnose_ZERO_Power_Fault_States
    (@PROPERTIES=
        Value    @TYPE=Boolean;
    )
)

(@OBJECT=    IFPC_Amplifier_Coupling_to_System
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=    IFPC_Amplifier_General_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=    IFPC_Amplifier_Power_Supply_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   IFPC_Attenuator_Coupling_to_System
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   IFPC_Attenuator_General_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   IFPC_Attenuator_Setting_Is_Incorrect
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
Matrix_Switch_Configuration_Is_Incorrect
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Matrix_Switch_General_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Matrix_Switch_Path_Setting_Is_Incorrect
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Undefined_General_Failure
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
ZZ_Outstanding_Fault_States_for_Matrix_Switch_Syst
em
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@SLOT= IFPC_Amplifier_Coupling_to_System.VERIFIED

@INFATOM=IFPC_Amplifier_Coupling_to_System.INF_CAT
EGORY;
)

(@SLOT= IFPC_Amplifier_General_Failure.VERIFIED

@INFATOM=IFPC_Amplifier_General_Failure.INF_CATEGO
RY;
)

(@SLOT=
IFPC_Amplifier_Power_Supply_Failure.VERIFIED

@INFATOM=IFPC_Amplifier_Power_Supply_Failure.INF_C
ATEGORY;
)

(@SLOT= IFPC_Attenuator_Coupling_to_System.VERIFIED

@INFATOM=IFPC_Attenuator_Coupling_to_System.INF_CA
TEGORY;
)

(@SLOT= IFPC_Attenuator_General_Failure.VERIFIED

@INFATOM=IFPC_Attenuator_General_Failure.INF_CATEG
ORY;
)

(@SLOT=
IFPC_Attenuator_Setting_Is_Incorrect.VERIFIED

@INFATOM=IFPC_Attenuator_Setting_Is_Incorrect.INF_
CATEGORY;
)

(@SLOT= Undefined_General_Failure.VERIFIED

@INFATOM=Undefined_General_Failure.INF_CATEGORY;
)
```

```
(@SLOT=
ZZ_Outstanding_Fault_States_for_Matrix_Switch_Syst
em.VERIFIED

@INFATOM=ZZ_Outstanding_Fault_States_for_Matrix_Sw
itch_System.INF_CATEGORY;
)

(@RULE= R1
    (@LHS=
        (Is (<|SUBSYSTEMS|>.POWER_LEVEL_OUT)
("HIGH"))
        (Yes
(<|HIGH_POWER_FAULT_STATES|>.VERIFIED))
    )
    (@HYPO= Diagnose_HIGH_Power_Fault_States)
)

(@RULE= R2
    (@LHS=
        (Is (<|SUBSYSTEMS|>.POWER_LEVEL_OUT)
("LOW"))
        (Yes
(<|LOW_POWER_FAULT_STATES|>.VERIFIED))
    )
    (@HYPO= Diagnose_LOW_Power_Fault_States)
)

(@RULE= R3
    (@LHS=
        (Is (<|SUBSYSTEMS|>.POWER_LEVEL_OUT)
("ZERO"))
        (Yes
(<|ZERO_POWER_FAULT_STATES|>.VERIFIED))
    )
    (@HYPO= Diagnose_ZERO_Power_Fault_States)
)
```

APPENDIX F
UP-CONVERTER SUBSYSTEMS DIAGNOSTIC KNOWLEDGE BASES

F.1 CHANNEL 1 UP-CONVERTER SUBSYSTEM

```
(@VERSION=  020)
(@PROPERTY= DESCRIPTION @TYPE=String;)
(@PROPERTY= INF_CATEGORY    @TYPE=Float;)
(@PROPERTY= POWER_LEVEL_OUT @TYPE=String;)
(@PROPERTY= VERIFIED    @TYPE=Boolean;)


(@CLASS=    FAULT_STATES
    (@SUBCLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@CLASS=    HIGH_POWER_FAULT_STATES
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@CLASS=    LOW_POWER_FAULT_STATES
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@CLASS=    SUBSYSTEMS
    (@PROPERTIES=
        POWER_LEVEL_OUT
    )
)

(@CLASS=    ZERO_POWER_FAULT_STATES
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)


(@OBJECT=   Diagnose_HIGH_Power_Fault_States
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@OBJECT=   Diagnose_LOW_Power_Fault_States
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@OBJECT=   Diagnose_ZERO_Power_Fault_States
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@OBJECT=   HPADIPC_Attenuator_Coupling_to_System
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPADIPC_Attenuator_General_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPADIPC_Attenuator_Setting_Is_Incorrect
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPAIPC_Amplifier_Coupling_to_System
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPAIPC_Amplifier_General_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPAIPC_Amplifier_Power_Supply_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPAIPC_Attenuator_Coupling_to_System
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPAIPC_Attenuator_General_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPAIPC_Attenuator_Setting_Is_Incorrect
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
Local_Oscillator_Frequency_Setting_Is_Incorrect
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Local_Oscillator_General_Failure
    (@CLASSES=
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
Local_Oscillator_Out_of_Phase_Lock__Alarm
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
Local_Oscillator_Out_of_Phase_Lock__Test
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Local_Oscillator_Power_Supply_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)
```

```
(@OBJECT=
Local_Oscillator_Signal_Power_Level_Is_LOW
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Mixer_Unit_Coupling_to_System
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Mixer_Unit_General_Failure
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Undefined_General_Failure
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
ZZ_Outstanding_Fault_States_for_Channel_1_Upconver
ter_System
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@SLOT=
HPADIPC_Attenuator_Coupling_to_System.VERIFIED

@INFATOM=HPADIPC_Attenuator_Coupling_to_System.INF
_CATEGORY;
)

(@SLOT=  HPADIPC_Attenuator_General_Failure.VERIFIED

@INFATOM=HPADIPC_Attenuator_General_Failure.INF_CA
TEGORY;
)

(@SLOT=
HPADIPC_Attenuator_Setting_Is_Incorrect.VERIFIED

@INFATOM=HPADIPC_Attenuator_Setting_Is_Incorrect.I
NF_CATEGORY;
)

(@SLOT=
HPAIPC_Amplifier_Coupling_to_System.VERIFIED

@INFATOM=HPAIPC_Amplifier_Coupling_to_System.INF_C
ATEGORY;
)

(@SLOT=  HPAIPC_Amplifier_General_Failure.VERIFIED

@INFATOM=HPAIPC_Amplifier_General_Failure.INF_CATE
GORY;
)

(@SLOT=
HPAIPC_Amplifier_Power_Supply_Failure.VERIFIED

@INFATOM=HPAIPC_Amplifier_Power_Supply_Failure.INF
_CATEGORY;
)

(@SLOT=
HPAIPC_Attenuator_Coupling_to_System.VERIFIED

@INFATOM=HPAIPC_Attenuator_Coupling_to_System.INF_
CATEGORY;
)

(@SLOT=  HPAIPC_Attenuator_General_Failure.VERIFIED

@INFATOM=HPAIPC_Attenuator_General_Failure.INF_CAT
EGORY;
)

(@SLOT=
HPAIPC_Attenuator_Setting_Is_Incorrect.VERIFIED

@INFATOM=HPAIPC_Attenuator_Setting_Is_Incorrect.IN
F_CATEGORY;
)

(@SLOT=
Local_Oscillator_Frequency_Setting_Is_Incorrect.VE
RIFIED

@INFATOM=Local_Oscillator_Frequency_Setting_Is_Inc
orrect.INF_CATEGORY;
)

(@SLOT=  Local_Oscillator_General_Failure.VERIFIED

@INFATOM=Local_Oscillator_General_Failure.INF_CATE
GORY;
)

(@SLOT=
Local_Oscillator_Out_of_Phase_Lock__Alarm.VERIFIED

@INFATOM=Local_Oscillator_Out_of_Phase_Lock__Alarm
.INF_CATEGORY;
)
```

```
(@SLOT=
Local_Oscillator_Out_of_Phase_Lock__Test.VERIFIED

@INFATOM=Local_Oscillator_Out_of_Phase_Lock__Test.
INF_CATEGORY;
)

(@SLOT=
Local_Oscillator_Power_Supply_Failure.VERIFIED

@INFATOM=Local_Oscillator_Power_Supply_Failure.INF
_CATEGORY;
)

(@SLOT=
Local_Oscillator_Signal_Power_Level_Is_LOW.VERIFIED

@INFATOM=Local_Oscillator_Signal_Power_Level_Is_LO
W.INF_CATEGORY;
)

(@SLOT= Mixer_Unit_Coupling_to_System.VERIFIED

@INFATOM=Mixer_Unit_Coupling_to_System.INF_CATEGOR
Y;
)

(@SLOT= Mixer_Unit_General_Failure.VERIFIED

@INFATOM=Mixer_Unit_General_Failure.INF_CATEGORY;
)

(@SLOT= Undefined_General_Failure.VERIFIED

@INFATOM=Undefined_General_Failure.INF_CATEGORY;
)

(@SLOT=
ZZ_Outstanding_Fault_States_for_Channel_1_Upconver
ter_System.VERIFIE\
D

@INFATOM=ZZ_Outstanding_Fault_States_for_Channel_1
_Upconverter_System.INF_CAT\
EGORY;
)

(@RULE= R1
    (@LHS=
        (Is (<|SUBSYSTEMS|>.POWER_LEVEL_OUT)
("HIGH"))
        (Yes
(<|HIGH_POWER_FAULT_STATES|>.VERIFIED))
    )
    (@HYPO= Diagnose_HIGH_Power_Fault_States)
)

(@RULE= R2
    (@LHS=
        (Is (<|SUBSYSTEMS|>.POWER_LEVEL_OUT)
("LOW"))
        (Yes
(<|LOW_POWER_FAULT_STATES|>.VERIFIED))
    )
    (@HYPO= Diagnose_LOW_Power_Fault_States)
)

(@RULE= R3
    (@LHS=
        (Is (<|SUBSYSTEMS|>.POWER_LEVEL_OUT)
("ZERO"))
        (Yes    (<|ZERO_POWER_FAULT_STATES|>.VERIFIED))
    )
    (@HYPO= Diagnose_ZERO_Power_Fault_States)
)
```

F.2 CHANNEL 2 UPCONVERTER SUBSYSTEM

```
(@VERSION=   020)
(@PROPERTY=  DESCRIPTION  @TYPE=String;)
(@PROPERTY=  INF_CATEGORY     @TYPE=Float;)
(@PROPERTY=  POWER_LEVEL_OUT  @TYPE=String;)
(@PROPERTY=  VERIFIED     @TYPE=Boolean;)


(@CLASS=     FAULT_STATES
    (@SUBCLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@CLASS=     HIGH_POWER_FAULT_STATES
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@CLASS=     LOW_POWER_FAULT_STATES
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@CLASS=     SUBSYSTEMS
    (@PROPERTIES=
        POWER_LEVEL_OUT
    )
)

(@CLASS=     ZERO_POWER_FAULT_STATES
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)


(@OBJECT=    Diagnose_HIGH_Power_Fault_States
    (@PROPERTIES=
        Value    @TYPE=Boolean;
    )
)

(@OBJECT=    Diagnose_LOW_Power_Fault_States
    (@PROPERTIES=
        Value    @TYPE=Boolean;
    )
)

(@OBJECT=    Diagnose_ZERO_Power_Fault_States
    (@PROPERTIES=
        Value    @TYPE=Boolean;
    )
)

(@OBJECT=    HPADIPC_Attenuator_Coupling_to_System
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=    HPADIPC_Attenuator_General_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=    HPADIPC_Attenuator_Setting_Is_Incorrect
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=    HPAIPC_Amplifier_Coupling_to_System
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=    HPAIPC_Amplifier_General_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=    HPAIPC_Amplifier_Power_Supply_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPAIPC_Attenuator_Coupling_to_System
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPAIPC_Attenuator_General_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPAIPC_Attenuator_Setting_Is_Incorrect
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
Local_Oscillator_Frequency_Setting_Is_Incorrect
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Local_Oscillator_General_Failure
    (@CLASSES=
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Local_Oscillator_Out_of_Phase_Lock__Alarm
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
Local_Oscillator_Out_of_Phase_Lock__Test
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Local_Oscillator_Power_Supply_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
Local_Oscillator_Signal_Power_Level_Is_LOW
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Mixer_Unit_Coupling_to_System
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Mixer_Unit_General_Failure
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Undefined_General_Failure
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
ZZ_Outstanding_Fault_States_for_Channel_2_Upconver
ter_System
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@SLOT=
HPADIPC_Attenuator_Coupling_to_System.VERIFIED

@INFATOM=HPADIPC_Attenuator_Coupling_to_System.INF
_CATEGORY;
)

(@SLOT= HPADIPC_Attenuator_General_Failure.VERIFIED

@INFATOM=HPADIPC_Attenuator_General_Failure.INF_CA
TEGORY;
)

(@SLOT=
HPADIPC_Attenuator_Setting_Is_Incorrect.VERIFIED

@INFATOM=HPADIPC_Attenuator_Setting_Is_Incorrect.I
NF_CATEGORY;
)

(@SLOT=
HPAIPC_Amplifier_Coupling_to_System.VERIFIED

@INFATOM=HPAIPC_Amplifier_Coupling_to_System.INF_C
ATEGORY;
)

(@SLOT= HPAIPC_Amplifier_General_Failure.VERIFIED

@INFATOM=HPAIPC_Amplifier_General_Failure.INF_CATE
GORY;
)

(@SLOT=
HPAIPC_Amplifier_Power_Supply_Failure.VERIFIED

@INFATOM=HPAIPC_Amplifier_Power_Supply_Failure.INF
_CATEGORY;
)

(@SLOT=
HPAIPC_Attenuator_Coupling_to_System.VERIFIED

@INFATOM=HPAIPC_Attenuator_Coupling_to_System.INF_
CATEGORY;
)

(@SLOT= HPAIPC_Attenuator_General_Failure.VERIFIED

@INFATOM=HPAIPC_Attenuator_General_Failure.INF_CAT
EGORY;
)

(@SLOT=
HPAIPC_Attenuator_Setting_Is_Incorrect.VERIFIED

@INFATOM=HPAIPC_Attenuator_Setting_Is_Incorrect.IN
F_CATEGORY;
)

(@SLOT=
Local_Oscillator_Frequency_Setting_Is_Incorrect.VE
RIFIED

@INFATOM=Local_Oscillator_Frequency_Setting_Is_Inc
orrect.INF_CATEGORY;
)

(@SLOT= Local_Oscillator_General_Failure.VERIFIED

@INFATOM=Local_Oscillator_General_Failure.INF_CATE
GORY;
)

(@SLOT=
Local_Oscillator_Out_of_Phase_Lock__Alarm.VERIFIED

@INFATOM=Local_Oscillator_Out_of_Phase_Lock__Alarm
.INF_CATEGORY;
)

(@SLOT=
Local_Oscillator_Out_of_Phase_Lock__Test.VERIFIED

@INFATOM=Local_Oscillator_Out_of_Phase_Lock__Test.
INF_CATEGORY;
)

(@SLOT=
Local_Oscillator_Power_Supply_Failure.VERIFIED

@INFATOM=Local_Oscillator_Power_Supply_Failure.INF
_CATEGORY;
)

(@SLOT=
Local_Oscillator_Signal_Power_Level_Is_LOW.VERIFIED

@INFATOM=Local_Oscillator_Signal_Power_Level_Is_LO
W.INF_CATEGORY;
)

(@SLOT= Mixer_Unit_Coupling_to_System.VERIFIED

@INFATOM=Mixer_Unit_Coupling_to_System.INF_CATEGOR
Y;
)

(@SLOT= Mixer_Unit_General_Failure.VERIFIED

@INFATOM=Mixer_Unit_General_Failure.INF_CATEGORY;
)

(@SLOT= Undefined_General_Failure.VERIFIED

@INFATOM=Undefined_General_Failure.INF_CATEGORY;
)
```

```
(@SLOT=
ZZ_Outstanding_Fault_States_for_Channel_2_Upconver
ter_System.VERIFIE\
D

@INFATOM=ZZ_Outstanding_Fault_States_for_Channel_2
_Upconverter_System.INF_CAT\
EGORY;
)

(@RULE= R1
    (@LHS=
        (Is (<¦SUBSYSTEMS¦>.POWER_LEVEL_OUT)
("HIGH"))
        (Yes
(<¦HIGH_POWER_FAULT_STATES¦>.VERIFIED))
    )
    (@HYPO= Diagnose_HIGH_Power_Fault_States)
)

(@RULE= R2
    (@LHS=
        (Is (<¦SUBSYSTEMS¦>.POWER_LEVEL_OUT)
("LOW"))
        (Yes
(<¦LOW_POWER_FAULT_STATES¦>.VERIFIED))
    )
    (@HYPO= Diagnose_LOW_Power_Fault_States)
)

(@RULE= R3
    (@LHS=
        (Is (<¦SUBSYSTEMS¦>.POWER_LEVEL_OUT)
("ZERO"))
        (Yes
(<¦ZERO_POWER_FAULT_STATES¦>.VERIFIED))
    )
    (@HYPO= Diagnose_ZERO_Power_Fault_States)
)
```

APPENDIX G
HIGH POWER AMPLIFIER SUBSYSTEMS DIAGNOSTIC KNOWLEDGE BASES

G.1 CHANNEL 1 AMPLIFIER SUBSYSTEM

```
(@VERSION=  020)
(@PROPERTY= DESCRIPTION @TYPE=String;)
(@PROPERTY= INF_CATEGORY    @TYPE=Float;)
(@PROPERTY= POWER_LEVEL_OUT @TYPE=String;)
(@PROPERTY= VERIFIED    @TYPE=Boolean;)


(@CLASS=    FAULT_STATES
    (@SUBCLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@CLASS=    HIGH_POWER_FAULT_STATES
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@CLASS=    LOW_POWER_FAULT_STATES
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@CLASS=    SUBSYSTEMS
    (@PROPERTIES=
        POWER_LEVEL_OUT
    )
)

(@CLASS=    ZERO_POWER_FAULT_STATES
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)


(@OBJECT=   Diagnose_HIGH_Power_Fault_States
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@OBJECT=   Diagnose_LOW_Power_Fault_States
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@OBJECT=   Diagnose_ZERO_Power_Fault_States
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@OBJECT=   HPADIPC_Attenuator_Coupling_to_System
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPADIPC_Attenuator_General_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPADIPC_Attenuator_Setting_Is_Incorrect
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPAIPC_Amplifier_Coupling_to_System
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPAIPC_Amplifier_General_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPAIPC_Amplifier_Power_Supply_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPAIPC_Attenuator_Coupling_to_System
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPAIPC_Attenuator_General_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPAIPC_Attenuator_Setting_Is_Incorrect
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
Local_Oscillator_Frequency_Setting_Is_Incorrect
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Local_Oscillator_General_Failure
    (@CLASSES=
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
Local_Oscillator_Out_of_Phase_Lock__Alarm
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
Local_Oscillator_Out_of_Phase_Lock__Test
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Local_Oscillator_Power_Supply_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)
```

```
(@OBJECT=
Local_Oscillator_Signal_Power_Level_Is_LOW
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Mixer_Unit_Coupling_to_System
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Mixer_Unit_General_Failure
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Undefined_General_Failure
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
ZZ_Outstanding_Fault_States_for_Channel_1_Upconver
ter_System
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@SLOT=
HPADIPC_Attenuator_Coupling_to_System.VERIFIED

@INFATOM=HPADIPC_Attenuator_Coupling_to_System.INF
_CATEGORY;
)

(@SLOT= HPADIPC_Attenuator_General_Failure.VERIFIED

@INFATOM=HPADIPC_Attenuator_General_Failure.INF_CA
TEGORY;
)

(@SLOT=
HPADIPC_Attenuator_Setting_Is_Incorrect.VERIFIED

@INFATOM=HPADIPC_Attenuator_Setting_Is_Incorrect.I
NF_CATEGORY;
)

(@SLOT=
HPAIPC_Amplifier_Coupling_to_System.VERIFIED

@INFATOM=HPAIPC_Amplifier_Coupling_to_System.INF_C
ATEGORY;
)

(@SLOT= HPAIPC_Amplifier_General_Failure.VERIFIED

@INFATOM=HPAIPC_Amplifier_General_Failure.INF_CATE
GORY;
)

(@SLOT=
HPAIPC_Amplifier_Power_Supply_Failure.VERIFIED

@INFATOM=HPAIPC_Amplifier_Power_Supply_Failure.INF
_CATEGORY;
)

(@SLOT=
HPAIPC_Attenuator_Coupling_to_System.VERIFIED

@INFATOM=HPAIPC_Attenuator_Coupling_to_System.INF_
CATEGORY;
)

(@SLOT= HPAIPC_Attenuator_General_Failure.VERIFIED

@INFATOM=HPAIPC_Attenuator_General_Failure.INF_CAT
EGORY;
)

(@SLOT=
HPAIPC_Attenuator_Setting_Is_Incorrect.VERIFIED

@INFATOM=HPAIPC_Attenuator_Setting_Is_Incorrect.IN
F_CATEGORY;
)

(@SLOT=
Local_Oscillator_Frequency_Setting_Is_Incorrect.VE
RIFIED

@INFATOM=Local_Oscillator_Frequency_Setting_Is_Inc
orrect.INF_CATEGORY;
)

(@SLOT= Local_Oscillator_General_Failure.VERIFIED

@INFATOM=Local_Oscillator_General_Failure.INF_CATE
GORY;
)

(@SLOT=
Local_Oscillator_Out_of_Phase_Lock__Alarm.VERIFIED

@INFATOM=Local_Oscillator_Out_of_Phase_Lock__Alarm
.INF_CATEGORY;
)
```

```
(@SLOT=
Local_Oscillator_Out_of_Phase_Lock__Test.VERIFIED

@INFATOM=Local_Oscillator_Out_of_Phase_Lock__Test.
INF_CATEGORY;
)

(@SLOT=
Local_Oscillator_Power_Supply_Failure.VERIFIED

@INFATOM=Local_Oscillator_Power_Supply_Failure.INF
_CATEGORY;
)

(@SLOT=
Local_Oscillator_Signal_Power_Level_Is_LOW.VERIFIED

@INFATOM=Local_Oscillator_Signal_Power_Level_Is_LO
W.INF_CATEGORY;
)

(@SLOT= Mixer_Unit_Coupling_to_System.VERIFIED

@INFATOM=Mixer_Unit_Coupling_to_System.INF_CATEGOR
Y;
)

(@SLOT= Mixer_Unit_General_Failure.VERIFIED

@INFATOM=Mixer_Unit_General_Failure.INF_CATEGORY;
)

(@SLOT= Undefined_General_Failure.VERIFIED

@INFATOM=Undefined_General_Failure.INF_CATEGORY;
)

(@SLOT=
ZZ_Outstanding_Fault_States_for_Channel_1_Upconver
ter_System.VERIFIE\
D

@INFATOM=ZZ_Outstanding_Fault_States_for_Channel_1
_Upconverter_System.INF_CAT\
EGORY;
)

(@RULE= R1
    (@LHS=
        (Is (<|SUBSYSTEMS|>.POWER_LEVEL_OUT)
("HIGH"))
        (Yes
(<|HIGH_POWER_FAULT_STATES|>.VERIFIED))
    )
    (@HYPO= Diagnose_HIGH_Power_Fault_States)
)

(@RULE= R2
    (@LHS=
        (Is (<|SUBSYSTEMS|>.POWER_LEVEL_OUT)
("LOW"))
        (Yes
(<|LOW_POWER_FAULT_STATES|>.VERIFIED))
    )
    (@HYPO= Diagnose_LOW_Power_Fault_States)
)

(@RULE= R3
    (@LHS=
        (Is (<|SUBSYSTEMS|>.POWER_LEVEL_OUT)
("ZERO"))
        (Yes    (<|ZERO_POWER_FAULT_STATES|>.VERIFIED))
    )
    (@HYPO= Diagnose_ZERO_Power_Fault_States)
)
```

G.2 CHANNEL 2 AMPLIFIER SUBSYSTEM

```
(@VERSION=  020)
(@PROPERTY= DESCRIPTION @TYPE=String;)
(@PROPERTY= INF_CATEGORY    @TYPE=Float;)
(@PROPERTY= POWER_LEVEL_OUT @TYPE=String;)
(@PROPERTY= VERIFIED    @TYPE=Boolean;)


(@CLASS=    FAULT_STATES
    (@SUBCLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@CLASS=    HIGH_POWER_FAULT_STATES
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@CLASS=    LOW_POWER_FAULT_STATES
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@CLASS=    SUBSYSTEMS
    (@PROPERTIES=
        POWER_LEVEL_OUT
    )
)

(@CLASS=    ZERO_POWER_FAULT_STATES
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)


(@OBJECT=   Diagnose_HIGH_Power_Fault_States
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@OBJECT=   Diagnose_LOW_Power_Fault_States
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@OBJECT=   Diagnose_ZERO_Power_Fault_States
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@OBJECT=   HPADIPC_Attenuator_Coupling_to_System
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPADIPC_Attenuator_General_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPADIPC_Attenuator_Setting_Is_Incorrect
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPAIPC_Amplifier_Coupling_to_System
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPAIPC_Amplifier_General_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPAIPC_Amplifier_Power_Supply_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPAIPC_Attenuator_Coupling_to_System
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPAIPC_Attenuator_General_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   HPAIPC_Attenuator_Setting_Is_Incorrect
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
Local_Oscillator_Frequency_Setting_Is_Incorrect
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Local_Oscillator_General_Failure
    (@CLASSES=
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
Local_Oscillator_Out_of_Phase_Lock__Alarm
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
Local_Oscillator_Out_of_Phase_Lock__Test
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Local_Oscillator_Power_Supply_Failure
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
Local_Oscillator_Signal_Power_Level_Is_LOW
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Mixer_Unit_Coupling_to_System
    (@CLASSES=
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=   Mixer_Unit_General_Failure
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)
```

```
(@OBJECT=   Undefined_General_Failure
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@OBJECT=
ZZ_Outstanding_Fault_States_for_Channel_2_Upconver
ter_System
    (@CLASSES=
        HIGH_POWER_FAULT_STATES
        LOW_POWER_FAULT_STATES
        ZERO_POWER_FAULT_STATES
    )
    (@PROPERTIES=
        DESCRIPTION
        INF_CATEGORY
        VERIFIED
    )
)

(@SLOT=
HPADIPC_Attenuator_Coupling_to_System.VERIFIED

@INFATOM=HPADIPC_Attenuator_Coupling_to_System.INF
_CATEGORY;
)

(@SLOT= HPADIPC_Attenuator_General_Failure.VERIFIED

@INFATOM=HPADIPC_Attenuator_General_Failure.INF_CA
TEGORY;
)

(@SLOT=
HPADIPC_Attenuator_Setting_Is_Incorrect.VERIFIED

@INFATOM=HPADIPC_Attenuator_Setting_Is_Incorrect.I
NF_CATEGORY;
)

(@SLOT=
HPAIPC_Amplifier_Coupling_to_System.VERIFIED

@INFATOM=HPAIPC_Amplifier_Coupling_to_System.INF_C
ATEGORY;
)

(@SLOT= HPAIPC_Amplifier_General_Failure.VERIFIED

@INFATOM=HPAIPC_Amplifier_General_Failure.INF_CATE
GORY;
)

(@SLOT=
HPAIPC_Amplifier_Power_Supply_Failure.VERIFIED

@INFATOM=HPAIPC_Amplifier_Power_Supply_Failure.INF
_CATEGORY;
)

(@SLOT=
HPAIPC_Attenuator_Coupling_to_System.VERIFIED

@INFATOM=HPAIPC_Attenuator_Coupling_to_System.INF_
CATEGORY;
)

(@SLOT= HPAIPC_Attenuator_General_Failure.VERIFIED

@INFATOM=HPAIPC_Attenuator_General_Failure.INF_CAT
EGORY;
)

(@SLOT=
HPAIPC_Attenuator_Setting_Is_Incorrect.VERIFIED

@INFATOM=HPAIPC_Attenuator_Setting_Is_Incorrect.IN
F_CATEGORY;
)

(@SLOT=
Local_Oscillator_Frequency_Setting_Is_Incorrect.VE
RIFIED

@INFATOM=Local_Oscillator_Frequency_Setting_Is_Inc
orrect.INF_CATEGORY;
)

(@SLOT= Local_Oscillator_General_Failure.VERIFIED

@INFATOM=Local_Oscillator_General_Failure.INF_CATE
GORY;
)

(@SLOT=
Local_Oscillator_Out_of_Phase_Lock__Alarm.VERIFIED

@INFATOM=Local_Oscillator_Out_of_Phase_Lock__Alarm
.INF_CATEGORY;
)

(@SLOT=
Local_Oscillator_Out_of_Phase_Lock__Test.VERIFIED

@INFATOM=Local_Oscillator_Out_of_Phase_Lock__Test.
INF_CATEGORY;
)

(@SLOT=
Local_Oscillator_Power_Supply_Failure.VERIFIED

@INFATOM=Local_Oscillator_Power_Supply_Failure.INF
_CATEGORY;
)

(@SLOT=
Local_Oscillator_Signal_Power_Level_Is_LOW.VERIFIED

@INFATOM=Local_Oscillator_Signal_Power_Level_Is_LO
W.INF_CATEGORY;
)

(@SLOT= Mixer_Unit_Coupling_to_System.VERIFIED

@INFATOM=Mixer_Unit_Coupling_to_System.INF_CATEGOR
Y;
)

(@SLOT= Mixer_Unit_General_Failure.VERIFIED

@INFATOM=Mixer_Unit_General_Failure.INF_CATEGORY;
)

(@SLOT= Undefined_General_Failure.VERIFIED

@INFATOM=Undefined_General_Failure.INF_CATEGORY;
)
```

```
(@SLOT=
ZZ_Outstanding_Fault_States_for_Channel_2_Upconver
ter_System.VERIFIE\
D

@INFATOM=ZZ_Outstanding_Fault_States_for_Channel_2
_Upconverter_System.INF_CAT\
EGORY;
)

(@RULE= R1
    (@LHS=
        (Is (<¦SUBSYSTEMS¦>.POWER_LEVEL_OUT)
("HIGH"))
        (Yes
(<¦HIGH_POWER_FAULT_STATES¦>.VERIFIED))
    )
    (@HYPO= Diagnose_HIGH_Power_Fault_States)
)

(@RULE= R2
    (@LHS=
        (Is (<¦SUBSYSTEMS¦>.POWER_LEVEL_OUT)
("LOW"))
        (Yes
(<¦LOW_POWER_FAULT_STATES¦>.VERIFIED))
    )
    (@HYPO= Diagnose_LOW_Power_Fault_States)
)

(@RULE= R3
    (@LHS=
        (Is (<¦SUBSYSTEMS¦>.POWER_LEVEL_OUT)
("ZERO"))
        (Yes
(<¦ZERO_POWER_FAULT_STATES¦>.VERIFIED))
    )
    (@HYPO= Diagnose_ZERO_Power_Fault_States)
)
```

APPENDIX H
SITE RELATED PUBLICATIONS

[1] **Bagwell, James W.,** *"A System for the Simulation and Evaluation of Satellite Communication Networks,"* 10th AIAA Communication Satellite Systems Conference, March 1984.

[2] **Budinger, James M.,** *"A Burst Compression and Expansion Technique for Variable Rate Users in Satellite-Switched TDMA Networks,"* 13th AIAA International Communication Satellite Systems Conference, March 1990.

[3] **Fujikawa, Gene, Conroy, Martin J., Saunders, Alan L., Pope, Dale E.,** *"Experimental Radio Frequency Link for Ka-Band Communications Applications,"* NASA TM, June 1988.

[4] **Fujikawa, Gene, Kerczewski, Robert J.,** *"Performance of a Ka-Band Satellite System Under Variable Signal Power Conditions,"* 1987 IEEE MTT Microwave Symposium, June 1987.

[5] **Gould, George R.,** *"A Modular Approach for Satellite Communication Ground Terminals,"* 10th AIAA Communication Satellite Systems Conference, January 1984.

[6] **Ivancic, William D., Andro, Monty, Nagy, Lawrence A., Budinger, James M., Shalkhauser, Mary Jo W.,** *"Satellite Matrix Switched Time Division Multiple Access Network Simulation and Evaluation,"* NASA TP, October 1989.

[7] **Ivancic, William D., Andro, Monty, Nagy, Lawrence A., Budinger, James M., Shalkhauser, Mary Jo W.,** *"Satellite Matrix Switched Time Division Multiple Access Network Simulation and Evaluation,"* 13th AIAA International Communication Satellite Systems Conference, March 1990.

[8] **Kerczewski, Robert J.,** *"A Study of the Effects of Group Delay Distortion of a SMSK Satellite Communications Channel,"* NASA TM, April 1987.

[9] **Kerczewski, Robert J.,** *"The Bit-Error Rate Performance of a Satellite Matrix Switch."* 12th AIAA Communication Satellite Systems Conference, March 1988.

[10] **Kerczewski, Robert J., Daugherty, Elaine S., Kramarchuk, Ihor,** *"Automated Measurement of the Bit-Error Rate as a Function of Signal-to-Noise Ratio for Microwave Communication Systems,"* 29th Automatic RF Techniques Group Conference, June 1987.

[11] **Kerczewski, Robert J., Daugherty, Elaine S., Kramarchuk, Ihor,** *"Gauge Bit-Error Rate as a Function of Signal-to-Noise Ratio,"* Microwaves and RF Magazine, March 1988.

[12] **Kerczewski, Robert J., Fujikawa, Gene,** *"Performance Measures for a Laboratory-Simulated 30/20 GHz Communication Satellite Transponder,"* 13th AIAA International Communication Satellite Systems Conference, March 1990.

[13] **Kerczewski, Robert J., Fujikawa, Gene, Svoboda, James A., Lizanich, Paul,** *"Effects of Amplitude Distortion and IF Equalization on Satellite Communication System Bit-Error Rate Performance,"* 13th AIAA International Communication Satellite Systems Conference, March 1990.

[14] **Kerczewski, Robert J., Ponchak, George E., Romanofsky, Robert R.,** *"Performance of Five 30 GHz Satellite Receivers,"* 1989 IEEE MTT International Microwave Symposium, June 1989.

[15] **Kerczewski, Robert J., Ponchak, George E., Romanofsky, Robert R.,** *"30 GHz Commercial Satellite Receivers,"* Applied Microwave Magazine, August 1989.

[16] **Leonard, Regis F., Kerczewski, Robert J.,** *"Radiofrequency Testing of Satellite Segment of Simulated 30/20 GHz Satellite Communication Systems,"* NASA TM, November 1985.

[17] **Nagy, Lawrence A.,** *"Satellite Range Delay Simulator for a Matrix Switched Time Division Multiple Access Network Simulation System,"* 13th AIAA International Communication Satellite Systems Conference, March 1990.

[18] **Shalkhauser, Kurt A., Fujikawa, Gene,** *"Bit-Error Rate Testing of High Power 30 GHz Traveling Wave Tube for Ground Terminal Applications,"* NASA TP, October 1986.

[19] **Shalkhauser, Kurt A., Kerczewski, Robert J.,** *"Automated Testing of Satellite Communication Systems,"* 25th Automatic Radiofrequency Techniques Group Conference, June 1985.

[20] **Shalkhauser, Kurt A., Nagy, Lawrence A., Svoboda, James S.,** *"Rein-Fade Simulation and Power Augmentation for Satellite Communication Systems,"* NASA TM 103134, September 1990.

[21] **Shalkhauser, Mary Jo W.,** *"Satellite Ground Terminal User Simulation,"* NASA TM, January 1988.

[22] **Shalkhauser, Mary Jo W.,** *"Design and Implementation of a Microcomputer-Based User Interface Controller for Bursted Data Communications Satellite Ground Terminals,"* NASA TM, December 1988.

[23] **Shalkhauser, Mary Jo W., Budinger, James M.,** *"Digitally Modulated Bit-Error-Rate Measurement System for Microwave Component Evaluation,"* NASA TP, July 1989.

[24] **Wald, Lawrence W.,** *"Characterization of a 30 GHz IMPATT Solid State Amplifier,"* NASA TM, July 1988.

[25] **Windmiller, Mary Jo,** *"Unique Bit-Error-Rate Measurement System for Satellite Communication Systems,"* NASA TP, March 1987.